

AD-A173 627

COMPUTER CENTER DEC VAXCLUSTER LIBRARIES/NSRDC
(SUBPROGRAMS)(U) DAVID M TAYLOR NAVAL SHIP RESEARCH AND
DEVELOPMENT CENTER BET D V SOMMER MAY 86

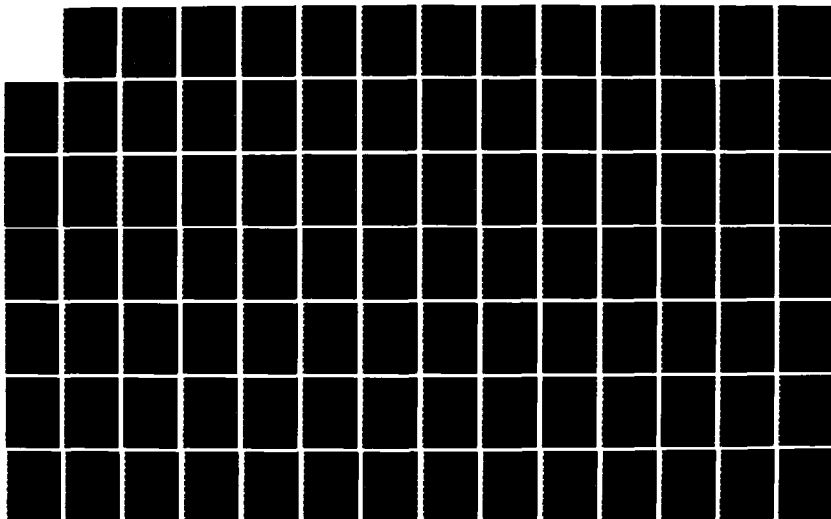
1/2

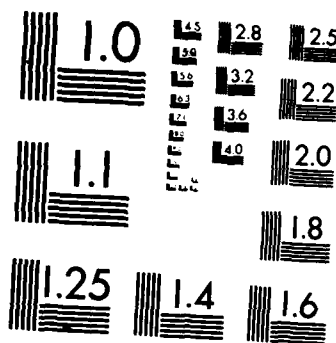
UNCLASSIFIED

DTNSRDC/CNLD-TM-18-86-13

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

12

TM-18-86-13

DAVID W. TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CENTER

Bethesda, Maryland 20084



COMPUTER CENTER
VAXCLUSTER LIBRARIES/NSRDC
(SUBPROGRAMS)

by

DAVID V. SOMMER

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

DTIC FILE COPY

Computation, Mathematics and Logistics Department
Technical Memorandum

DTIC
ELECTRONIC
OCT 20 1986
S E D

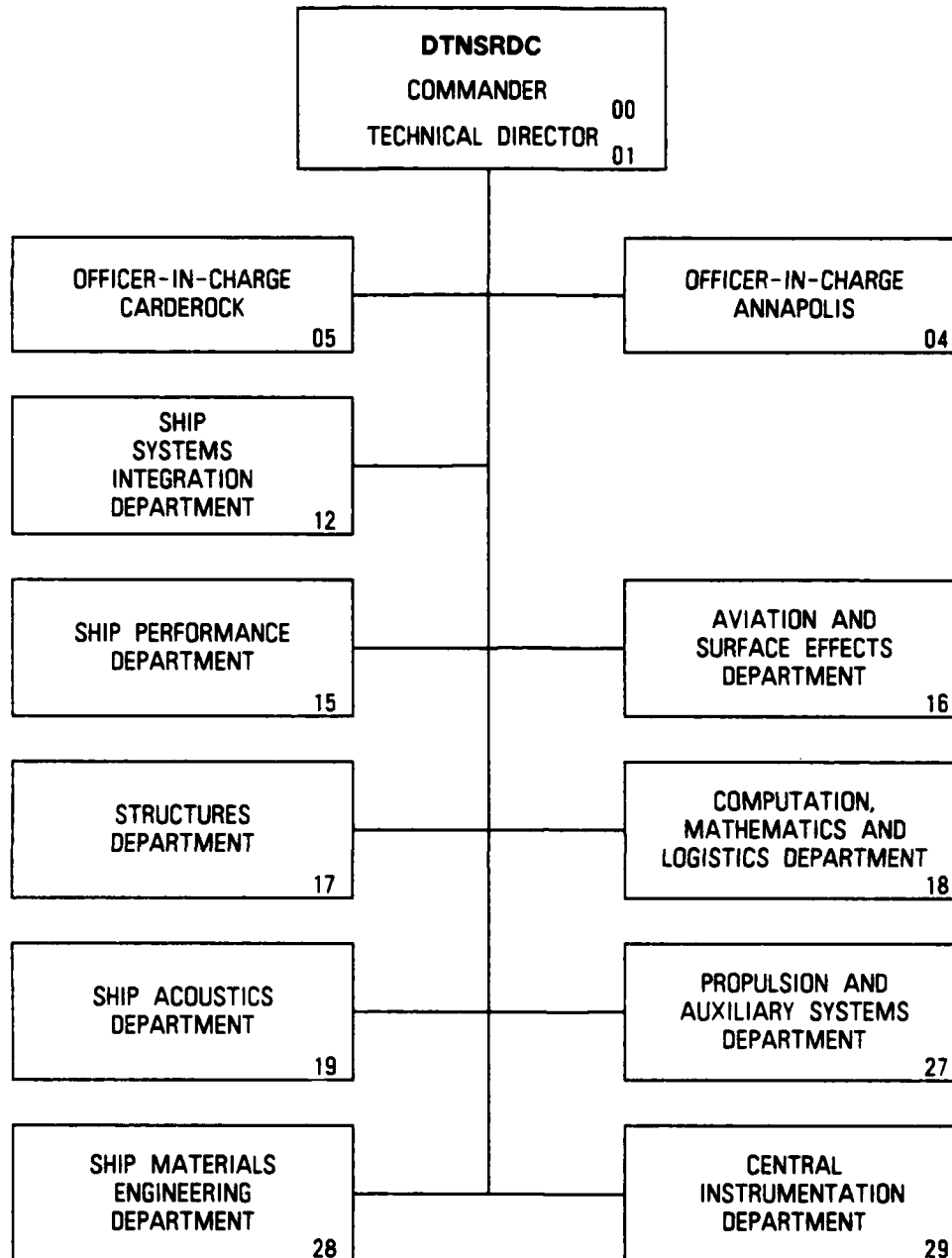
May 1986

TM-18-86-13

AD-A173 627

Computer Center VAXcluster Libraries/NSRDC (Subprograms)

MAJOR DTNSRDC ORGANIZATIONAL COMPONENTS



REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM								
1. REPORT NUMBER TM-18-86-13	2. GOVT ACCESSION NO AD-A173627	3. RECIPIENT'S CATALOG NUMBER								
4. TITLE (and Subtitle) Computer Center DEC VAXcluster Libraries: NSRDC (Subprograms)		5. TYPE OF REPORT & PERIOD COVERED								
		6. PERFORMING ORG. REPORT NUMBER Final								
7. AUTHOR(s) David V. Sommer		8. CONTRACT OR GRANT NUMBER(s)								
9. PERFORMING ORGANIZATION NAME AND ADDRESS DTNSRDC, User Services, Code 1892 Bethesda, Maryland 20084-5000		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS								
11. CONTROLLING OFFICE NAME AND ADDRESS Computation, Mathematics & Logistics Dept. Computer Facilities Division (189)		12. REPORT DATE May 1986								
		13. NUMBER OF PAGES 121								
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified								
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE								
16. DISTRIBUTION STATEMENT (of this Report) Approved for Public Release; Distribution Unlimited										
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)										
18. SUPPLEMENTARY NOTES										
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) <table border="0"> <tr> <td>Computers</td> <td>Libraries</td> </tr> <tr> <td>Control statements</td> <td>Software documentation</td> </tr> <tr> <td>DEC VAXcluster</td> <td>Subprograms</td> </tr> <tr> <td>Functional categories</td> <td></td> </tr> </table>			Computers	Libraries	Control statements	Software documentation	DEC VAXcluster	Subprograms	Functional categories	
Computers	Libraries									
Control statements	Software documentation									
DEC VAXcluster	Subprograms									
Functional categories										
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) <p>The Computer Center DEC VAXcluster Libraries: NSRDC (Subprograms), VLIB/N is a reference manual which describes the subprograms in library VSYS:NSRDC on the DEC VAXcluster at DTNSRDC. These scientific and utility routines are used primarily with Fortran programs and are written in Fortran. VLIB/N lists the on-line helps and includes a list by functional category and an alphabetical list with a descriptive title for each.</p>										

David W. Taylor
Naval Ship Research and Development Center
Bethesda, Maryland 20084-5000

*
*
* Computer Center *
* DEC VAXcluster *
* Libraries: NSRDC *
* (Subprograms) *
*
*

by
David V. Sommer

User Services Branch
Code 1892

	Carderock	Annapolis
Phone	(202) 227-1907	(301) 267-3343
Autovon	287-1907	281-3343



Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

For recorded message on computer status (202) 227-3043

Questions and requests for more detailed information
should be directed to Code 1892, Bldg. 17, Rm. 100.

Computation, Mathematics and Logistics Department
Technical Memorandum

May 1986

TM-18-86-13

Table of Contents

1	Introduction	
	What's In This Manual	1-1
	Contents	1-2
	Functional Categories	1-6
	List of Routines by Category	1-9
2	Routine documentation	
	How to Get On line Help	2-1
	<individual HELP modules arranged alphabetically>	2-2 *

*** How This Was Prepared ***

This is a printed document of the on-line help modules available. There has been no attempt to "neaten" them up -- the spacing is as it was designed to be displayed by the VMS HELP program. A procedure and a program were written to extract, arrange and format them.

* - As new routines are developed, the HELP modules may be printed and inserted into this document.

***** Introduction *****

The Computer Center makes available on VAXcluster, in addition to the VMS operating system, a wide variety of both scientific and utility programs, subprograms and procedures. The routines are maintained in libraries or as separate files in the VSYS: directory.

The VLIB-Series consists of the following, which are the helps for the various VAXcluster "libraries" maintained by the Computer Center:

VLIB/D - Computer Center VAXcluster Libraries / DTNSRDC (Commands and General Information)	TM-18-86-12
VLIB/N - Computer Center VAXcluster Libraries / NSRDC (Subprograms)	TM-18-86-13
VLIB/P - Computer Center VAXcluster Libraries / PROCFIL (Procedures)	TM-18-86-14
VLIB/U - Computer Center VAXcluster Libraries / UTILITY (Programs)	TM-18-86-15

*** What's In This Manual ***

A list of the routines with a brief description of each is followed by the list of functional categories used to classify each routine. Next is a list of the routines under the various categories. Chapter 2 contains the currently available HELP modules in alphabetical order.

**** Contents ****

The following subprograms were written at DTNSRDC and are in object library VSYS:NSRDC.OLB. For help, type "HELP @NSRDC routine".

AC	Character function to get current job order number.
ALFA	Test character for alphabetic.
ALFANU	Test character for alphanumeric.
ALFANUS	Test character string for alphanumeric.
ALFAS	Test character string for alphabetic.
BANR	Write a banner (characters are 10 lines high; lines are 110 positions wide).
BANR6	Write a banner (characters are 6 lines high; lines are 80 positions wide).
BITPKG	A package of four subprograms to give high-level language access to large bit arrays.
ByCategory	List of modules by the functional category to which each belongs.
ByDate	List of modules in reverse order by the date of the last modification to the module or its help.
C2VDAT	Convert CDC format date (mm/dd/yy) to VMS format (dd-mmm-yy).
CENTER	Integer function to center a character string. The string is centered within itself.
CHIN	Integer function to convert a numeric character string to an integer.
CLRBIT	Clear one bit in a bit array.
CPU	Get the CPU processor for this node.
CSHUFL	Shuffle a character array.
CSORT	Sort (ascending) a character array.
CSORT2	Sort (ascending) a character array having an associated character array.
CSORT2D	Sort (descending) a character array having an associated character array.
CSORTD	Sort (descending) a character array.
CSORTN	Sort (ascending) a character array having an associated non-character array.

CSORTND Sort (descending) a character array having an associated non-character array.

DIGIT Test character for digit.

DIGITS Test character string for digit.

FLPBIT Flip one bit in a bit array.

FRSTCH Integer function to return the position of the first non-blank in a character string.

GETSTR Extract character string according to user-defined criteria.

HMS2S Convert hh:mm:ss to seconds.

IOSTAT_TEXT Convert the Fortran I/O status code to a message.

ISORTC Sort (ascending) an integer array having an associated character array.

ISORTCD Sort (descending) an integer array having an associated character array.

ISUM Sum an integer array.

ISVT100 Determine if output file (SYSS\$OUTPUT) is VT-100-compatible.

ITRANS Integer function to translate characters according to translate tables you specify in the call.

JGDATE Convert any Gregorian date to a relative Julian number or vice versa.

JPMODE Get the job/process mode (batch, interactive, network, other, or unknown).

LEFT Integer function to left-justify a character string. The string is left-justified within itself.

LO2UP Convert lower case to upper case.

LOWER Test character for lower case letter.

LSTCH Integer function to return the position of the last non-blank in a character string.

MAXAI Find the maximum of an array of integers.

MAXAR Find the maximum of an array of real numbers.

MAXINT Return the maximum integer supported by VAX/VMS.

MAXREAL Return the maximum real number supported by VAX/VMS.

MFRAME Obtain the machine and node running the program.

MINAI Find the minimum of an array of integers.

MINAR Find the minimum of an array of real numbers.

MININT Return the minimum integer supported by VAX/VMS.

MINREAL Return the minimum real number (absolute value) supported by VAX/VMS.

MOVEIT Move a real or integer array.

NARGS Determine the number of arguments with which a subprogram was called.

NEWFILETYPE Replace filetype (and version) of a filespec.

PARS Parse a string.

PARSEFILESPEC Parse a file specification, that is, break it up into its components.

QUALCHAR Extract string from character qualifier.

QUALINT Extract string from integer qualifier.

QUALLOG Extract string from logical qualifier.

REPLAC Replace characters in a string with a character.

REPLEQ Replace characters in a string with other characters.

REPLNE Replace unspecified characters in a string with a character.

REVERSE Reverse the order of characters in a character string.

RIGHT Integer function to right-justify a character string. The string is right-justified within itself.

S2HMS Convert seconds to hh:mm:ss.

SETBIT Set one bit in a bit array.

SIGDIG Return number of significant digits (including 1 for a minus sign, if needed)

SUM Sum a real array.

SWAPCASE Swap lower and upper case.

SY Solve tridiagonal system of equations following the Thomas algorithm.

TERMINAL For interactive users, get the terminal name.

TRANS Translate characters according to translate tables you specify in the call.

TSTARGDFT In a subprogram, test whether a specific argument in the call

exists and is not defaulted.

TSTBIT	Test one bit in a bit array.
UP2LO	Convert upper case to lower case.
UPPER	Test character for upper case letter.
USERID	Get user initials.
V2CDAT	Convert VMS format date (dd-mmm-yy) to CDC format (mm/dd/yy).
WEKDAY	Find the day-of-the-week.

*** Functional Categories ***

The following functional categories are used at DTNSRDC. Those preceded by an asterisk (*) are local DTNSRDC categories. All others are from VIM (the CDC users group).

- A0 Arithmetic routines
 - A1 Real numbers
 - A2 Complex numbers
 - A3 Decimal
 - A4 I/O routines
- B0 Elementary functions
 - B1 Trigonometric
 - B2 Hyperbolic
 - B3 Exponential and logarithmic
 - B4 Roots and powers
- C0 Polynomials and special functions
 - C1 Evaluation of polynomials
 - C2 Roots of polynomials
 - C3 Evaluation of special functions (non-statistical)
 - C4 Simultaneous non-linear algebraic equations
 - C5 Simultaneous transcendental equations
 - * C6 Roots of functions
- D0 Operations on functions and solutions of differential equations
 - D1 Numerical integration
 - D2 Numerical solutions of ordinary differential equations
 - D3 Numerical solutions of partial differential equations
 - D4 Numerical differentiation
- E0 Interpolation and approximations
 - E1 Table look-up and interpolation
 - E2 Curve fitting
 - E3 Smoothing
 - E4 Minimizing or maximizing a function
- F0 Operations on matrices, vectors & simultaneous linear equations
 - F1 Vector and matrix operations
 - F2 Eigenvalues and eigenvectors
 - F3 Determinants
 - F4 Simultaneous linear equations
- G0 Statistical analysis and probability
 - G1 Data reduction (common statistical parameters)
 - G2 Correlation and regression analysis
 - G3 Sequential analysis
 - G4 Analysis of variance
 - G5 Time series
 - G6 Special functions (includes random numbers and pdf's)
 - * G7 Multivariate analysis and scale statistics
 - * G8 Non-parametric methods and statistical tests
 - * G9 Statistical inference

- H0 Operations research techniques, simulation & management science
 - H1 Linear programming
 - H2 Non-linear programming
 - H3 Transportation and network codes
 - H4 Simulation modeling
 - H5 Simulation models
 - H6 Critical path programs
 - H8 Auxiliary programs
 - H9 Combined
- I0 Input
 - I1 Binary
 - I2 Octal
 - I3 Decimal
 - I4 BCD (Hollerith)
 - I9 Composite
- J0 Output
 - J1 Binary
 - J2 Octal
 - J3 Decimal
 - J4 BCD (Hollerith)
 - J5 Plotting
 - J7 Analog
 - J9 Composite
- K0 Internal information transfer
 - K1 External-to-external
 - K2 Internal-to-internal (relocation)
 - K3 Disk
 - K4 Tape
 - K5 Direct data devices
- L0 Executive routines
 - L1 Assembly
 - L2 Compiling
 - L3 Monitoring
 - L4 Preprocessing
 - L5 Disassembly and derelativizing
 - L6 Relativizing
 - L7 Computer language translators
- M0 Data handling
 - M1 Sorting
 - M2 Conversion and/or scaling
 - M3 Merging
 - M4 Character manipulation
 - M5 Searching, seeking, locating
 - M6 Report generators
 - M9 Composite
- N0 Debugging
 - N1 Tracing and trapping
 - N2 Dumping
 - N3 Memory verification and searching
 - N4 Breakpoint printing

- 00 Simulation of computers and data processors (interpreters)
 - 01 Off-line equipment (listers, reproducers, etc.)
 - 03 Computers
 - 04 Pseudo-computers
 - 05 Software simulation of peripherals
 - 09 Composite
- P0 Diagnostics (hardware malfunction)
- Q0 Service or housekeeping, programming aids
 - Q1 Clear/reset
 - Q2 Checksum accumulation and correction
 - Q3 File manipulation
 - Q4 Internal housekeeping, save, restore, etc.
 - Q5 Report generator subroutines
 - Q6 Program documentation: flow charts, document standardization
 - Q7 Program library utilities
- R0 Logic and symbolic
 - R1 Formal logic
 - R2 Symbol manipulation
 - R3 List and string processing
 - R4 Text editing
- S0 Information retrieval
- T0 Applications and application-oriented programs
 - T1 Physics (including nuclear)
 - T2 Chemistry
 - T3 Other physical sciences (geology, astronomy, etc.)
 - T4 Engineering
 - T5 Business data processing
 - T6 Manufacturing (non-data) processing and process control
 - T7 Mathematics and applied mathematics
 - T8 Social and behavioral sciences and psychology
 - T9 Biological sciences
 - T10 Regional sciences (geography, urban planning)
 - T11 Computer assisted instruction
- U0 Linguistics and languages
- V0 General purpose utility subroutines
 - V1 Random number generators
 - V2 Combinatorial generators: permutations, combinations & subsets
 - * V3 standard and special problems
- X0 Data reduction
 - X1 Re-formatting, decommutation, error diagnosis
 - X2 Editing
 - X3 Calibration
 - X4 Evaluation
 - X5 Analysis (time-series analysis)
 - X6 Simulation (generate test data for data reduction system)
- Y0 Installation modification
 - Y1 Installation modification library
 - Y2 NEWPL tape of installation modifications
- Z0 All others

**** By Functional Category ****
 (13-JUN-86 @ 08:39:22)

The modules in this library are listed below by functional category.

(E - executable program; F - function subprogram; P - procedure;
 S - subroutine subprogram; Z - miscellaneous)

A1 Real numbers

F-ISUM F-SUM

F4 Simultaneous linear equations

S-SY

J4 BCD (Hollerith)

S-BANR S-BANR6

M1 Sorting

S-CSHUFL S-CSORT S-CSORT2 S-CSORT2D S-CSORTD
 S-CSORTN S-ISORTC S-ISORTCD

M2 Conversion and/or scaling

S-C2VDAT S-CHIN S-HMS2S S-JGDATE S-LO2UP
 S-S2HMS S-UP2LO S-V2CDAT

M4 Character manipulation

S-ALFA S-ALFANU S-ALFANUS S-ALFAS Z-BIT_PKG
 S-CENTER S-CLR_BIT S-DIGIT S-DIGITS S-FLP_BIT
 F-GETSTR S-ITRANS S-LEFT S-LOWER S-MOVEIT
 S-NEWFILETYP F-QUAL_CHAR F-QUAL_INT F-QUAL_LOG F-REPLAC
 F-REPLEQ F-REPLNE S-REVERSE S-RIGHT S-SET_BIT
 S-SWAPCASE S-TRANS F-TST_ARG_DF F-TST_BIT S-UPPER

M5 Searching, seeking, locating

F-FRSTCH F-GETSTR F-LSTCH F-MAXAI F-MAXAR
 F-MAXINT F-MAXREAL F-MINAI F-MINAR F-MININT
 F-MINREAL F-PARS S-PARSE_FILE

Q0 Service or housekeeping, programming aids

F-AC F-CPU F-IS_VT100 F-JP_MODE S-MFRAME
 F-NARGS S-SIGDIG F-TERMINAL S-USERID S-WEKDAY

Q3 File manipulation

S-IOSTAT_TEX

***** Individual Documents *****

This chapter contains the HELP modules for all routines and general information in "library" NSRDC.

For the most recent on-line HELPs, type

HELP @NSRDC <routine>

To see the current contents, type

HELP @NSRDC Contents

To see the most recently changed routines of HELPs, type

HELP @NSRDC By_Date

To see the current functional category list of the modules, type

HELP @NSRDC By_Category

**** AC ****

Character function to get the current job order number.

Usage: CHARACTER AC * 10, JON * 10

...
JON = AC ()

*** Parameters ***

AC ()

AC - out - ch*10 - will contain the current job order number

*** Example ***

AC ()

CHARACTER AC * 10, JON * 10
INTEGER CHIN, NUMBER

...
JON = AC ()
PRINT *, 'The current job order number is', JON, '.'

**** ALFA ****

Test a character for alphabetic.

Usage: CHARACTER * 1 CH
LOGICAL ALFA
...
IF (ALFA(CH)) THEN
...

*** Parameters ***

ALFA (CH)

CH - in - ch*1 - character to be tested
ALFA - out - log - TRUE - CH is alphabetic
FALSE - CH is not alphabetic

*** Example ***

ALFA (CH)

Read a character string and flag all alphabetic characters.

```
...
CHARACTER STRING * 50, FLAGS * 50
...
FLAGS = ' '
READ (*, '(A)') STRING
DO 110 N=1,50
  IF (ALFA (STRING(N:N))) FLAGS(N:N) = '^'
110 CONTINUE
PRINT *, STRING
PRINT *, FLAGS
```

Then, for string='abcde FGHIJ kLmnO pQRst UvWxy Z1234567890()' \$
flags = '^^^^^ ^^^^^ ^^^^^ ^^^^^ ^^^^^ ^

**** ALFANU ****

Test a character for alphanumeric.

Usage: CHARACTER * 1 CH
 LOGICAL ALFANU
 ...
 IF (ALFANU(CH)) THEN
 ...

*** Parameters ***

ALFANU (CH)

CH - in - ch*1 - character to be tested

ALFANU - out - log - TRUE - CH is alphanumeric
 FALSE - CH is not alphanumeric

*** Example ***

ALFANU (CH)

Read a character string and flag all alphanumeric characters.

```

...
CHARACTER STRING * 50, FLAGS * 50
...
FLAGS = ' '
READ (*, '(A)') STRING
DO 110 N=1,50
  IF (ALFANU (STRING(N:N))) FLAGS(N:N) = '^'
110 CONTINUE
PRINT *, STRING
PRINT *, FLAGS

```

Then, for string='abcde FGHIJ kLmnO pQRst UvWxy Z1234567890()'S
 flags = 'AAAAA AAAAA AAAAA AAAAA AAAAA AAAAAA'

**** ALFANUS ****

Test a character string for alphanumeric.

Usage: CHARACTER * (N) STRING
 LOGICAL ALFANUS
 ...
 IF (ALFANUS(STRING)) THEN
 ...

*** Parameters ***

ALFANUS (STRING)

STRING - in - ch** - string to be tested

ALFANUS - out - log - TRUE - string was alphanumeric
 FALSE - string was not alphanumeric

*** Example ***

ALFANUS (STRING)

Read a character string and test for all alphanumeric.

```

...
CHARACTER STRING * 10
...
READ (*, '(A)') STRING
IF (ALFANUS (STRING(N:N))) THEN
  PRINT *, 'The string is all alphanumeric.'
ELSE
  PRINT *, 'The string has at least one non-alphanumeric character.'
END IF
...

```

Then for STRING='ab3defg8ij', this program segment prints:
 The string is all alphanumeric.

For STRING='abcde6*hij', this program segment prints:
 The string has at least one non-alphanumeric character.

**** ALFAS ****

Test a character string for alphabetic.

Usage: CHARACTER * (N) STRING
LOGICAL ALFAS

...
IF (ALFAS(STRING)) THEN
...

*** Parameters ***

ALFAS (STRING)

STRING - in - ch** - string to be tested

ALFAS - out - log - TRUE - string was alphabetic
FALSE - string was not alphabetic

*** Example ***

ALFAS (STRING)

Read a character string and test for all alphabetic.

...
CHARACTER STRING * 10
...
READ (*, '(A)') STRING
IF (ALFAS (STRING(N:N))) THEN
PRINT *, 'The string is all alphabetic.'
ELSE
PRINT *, 'The string has at least one non-alphabetic character.'
END IF
...

Then for STRING='abcdefghij', this program segment prints:
The string is all alphabetic.

For STRING='abcde6ghij', this program segment prints:
The string has at least one non-alphabetic character.

**** BANR ****

Write a banner (characters are 10 lines high; lines are 110 positions wide).

Usage: INTEGER OUTPUT_UNIT, WHERE_ON_PAGE

CALL BANR ('message', OUTPUT_UNIT, WHERE_ON_PAGE)

*** Parameters ***

CALL BANR ('message', OUTPUT_UNIT, WHERE_ON_PAGE)

MESSAGE - in - ch** - string to be printed
(10 characters maximum)

OUTPUT_UNIT - in - int - unit number for output
(for standard output file, use -1)

WHERE_ON_PAGE - in - int - 0 - put banner on new page
<>0 - put banner on same page

At present, BANR supports only the CDC 63-character set:

ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+ - * / () \$ % , . # [] : " _ ! & ' ? < > @ \ ^ ;

*** Example ***

CALL BANR ('message', OUTPUT_UNIT, WHERE_ON_PAGE)

Write a 2-line banner page with SHIP #
<ship number>
on the standard output file.

CHARACTER SHIPNO * 10

READ '(A)', SHIPNO

CALL BANR ('SHIP #', -1, 0)

CALL BANR (SHIPNO, -1, 1)

**** BANR6 ****

Write a banner (characters are 6 lines high; lines are 80 positions wide).

Usage: INTEGER OUTPUT_UNIT, WHERE_ON_PAGE

CALL BANR6 ('message', OUTPUT_UNIT, WHERE_ON_PAGE)

*** Parameters ***

CALL BANR6 ('message', OUTPUT_UNIT, WHERE_ON_PAGE)

MESSAGE - in - ch** - string to be printed
(10 characters maximum)

OUTPUT_UNIT - in - int - unit number for output
(for standard output file, use -1)

WHERE_ON_PAGE - in - int - 0 - put banner on new page
<>0 - put banner on same page

At present, BANR supports only the CDC 63-character set:

ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+~*/()\$= ,.#[]:"'_!&'?<>@\^;

*** Example ***

CALL BANR6 ('message', OUTPUT_UNIT, WHERE_ON_PAGE)

Write a 2-line banner page with SHIP #
<ship number>
on the standard output file.

CHARACTER SHIPNO * 10

READ '(A)', SHIPNO

CALL BANR6 ('SHIP #', -1, 0)

CALL BANR6 (SHIPNO, -1, 1)

***** BIT_PKG *****

This package provides high-level language access to large bit arrays.

It provides for setting, clearing, flipping, and testing individual bits in a bit array or string.

*** CLR_BIT ***

Clear one bit in a bit array (bit string).

Usage: CALL CLR_BIT (BITNO.rl.r, BITS.mv.r)

** Parameters **

CALL CLR_BIT (BITNO.rl.r, BITS.mv.r)

BITNO - in - int - the number of the bit to be cleared

BITS - i/o - - the bit string or array

** Example **

Clear bit 76 in a 100-bit table:

```

INTEGER N_BITS, BITS_WORD, N_WORDS
PARAMETER ( BITS_WORD = 32 ! integer*4 word
N          , N_BITS    = 100 ! in bit array
N          , N_WORDS   = (N_BITS + BITS_WORD - 1) / BITS_WORD
)
INTEGER BITNO, TABLE(N_WORDS)
...
BITNO = 76
CALL CLR_BIT (BITNO, TABLE)

```

*** FLP_BIT ***

Flip one bit in a bit array (bit string).

Usage: CALL FLP_BIT (BITNO.rl.r, BITS.mv.r)

** Parameters **

CALL FLP_BIT (BITNO.rl.r, BITS.mv.r)

BITNO - in - int - the number of the bit to be flipped

BITS - i/o - the bit string or array

**** Example ****

Flip bit 76 in a 100-bit table:

```
integer n_bits, bits_word, n_words
PARAMETER ( BITS_WORD = 32      ! integer*4 word
N           , N_BITS    = 100    ! bit array
N           , N_WORDS   = (N_BITS + BITS_WORD - 1) / BITS_WORD
)
INTEGER BITNO, TABLE(N_WORDS)
...
BITNO = 76
CALL FLP_BIT (BITNO, TABLE)
```

***** SET_BIT *****

Set one bit in a bit array (bit string).

Usage: CALL SET_BIT (BITNO.rl.r, BITS.mv.r)

**** Parameters ****

CALL SET_BIT (BITNO.rl.r, BITS.mv.r)

BITNO - in - int - the number of the bit to be set

BITS - i/o - the bit string or array

**** Example ****

Set bit 76 in a 100-bit table:

```
integer n_bits, bits_word, n_words
PARAMETER ( BITS_WORD = 32      ' integer*4 word
N           , N_BITS    = 100    in bit array
N           , N_WORDS   = (N_BITS + BITS_WORD - 1) / BITS_WORD
)
INTEGER BITNO, TABLE(N_WORDS)
...
BITNO = 76
CALL SET_BIT (BITNO, TABLE)
```

***** TST_BIT *****

Test one bit in a bit array (bit string).

Usage: LOGICAL BIT_SET, TST_BIT
 BIT_SET = TST_BIT (BITNO.rl.r, BITS.mv.r)

** Parameters **

TST_BIT (BITNO.rl.r, BITS.mv.r)

BITNO - in - int - the number of the bit to be tested

BITS - i/o - the bit string or array

TST_BIT - out - log - TRUE - the bit is set
 FALSE - the bit is not set

** Example **

Test bit 76 in a 100-bit table and print a message:

```

INTEGER N_BITS, BITS_WORD, N_WORDS
PARAMETER ( BITS_WORD = 32      ! integer*4 word
N           , N_BITS   = 100    ! in bit array
N           , N_WORDS  = (N_BITS + BITS_WORD - 1) / BITS_WORD
)
INTEGER BITNO, TABLE(N_WORDS)
LOGICAL TST_BIT
...
BITNO = 76
IF (TST_BIT (BITNO, TABLE)) THEN
  PRINT *, 'Bit ', BITNO, ' is set.'
ELSE
  PRINT *, 'Bit ', BITNO, ' is not set.'
END IF

```

*** Admin_info ***

Authors: F. Nagy - Fermilab Accelerator Control System (clr_bit,
 set_bit, tst_bit)
 David V. Sommer - DTNSRDC Code 1892.2 (flp_bit)

Languages: MACRO (clr_bit, set_bit, tst_bit)
 Fortran 77 (flp_bit)

Date written: 01/17/83 (clr_bit, set_bit, tst_bit)
 08/30/85 (flp_bit)

Dates revised

86/05/30

VAX

NSRDC

BIT_PKG

Page 2-12

**** C2VDAT ****

Convert CDC format date (mm/dd/yy) to VMS format (dd-mmm-yy).

Usage: CHARACTER CDC * 8, VMS * 9

CALL C2VDAT (CDC, VMS)

*** Parameters ***

CALL C2VDAT (CDC, VMS)

CDC - in - ch*8 - CDC format date to be converted (mm/dd/yy)

VMS - out - ch*9 - VMS format converted date (dd-mmm-yy)

*** Example ***

CALL C2VDAT (CDC, VMS)

CHARACTER CDC * 8, VMS * 9

CDC = '04/11/85'

CALL C2VDAT (CDC, VMS)

TYPE *, 'CDC date is ', CDC

TYPE *, 'VMS date is ', VMS

results in the following output:

CDC date is 04/11/85

VMS date is 11-APR-85

**** CENTER ****

Integer function to center a character string. The string is centered within itself.

Usage: CHARACTER STRING * (n)
 CHARACTER WORK * (n)
 INTEGER CENTER, LSTRING
 ...
 LSTRING = CENTER (STRING, WORK)

*** Parameters ***

CENTER (STRING, WORK)

STRING - i/o - ch** - string to be centered

WORK - - ch** - work variable of len(string)

CENTER - out - int - the position of the last non-blank

*** Example ***

CENTER (STRING, WORK)

CHARACTER LINE * 20
 CHARACTER WORK * 20
 INTEGER CENTER, LLINE

...
 READ '(A)', LINE
 LLINE = CENTER (LINE, WORK)

If LINE contains 'Some words', then after centering, LINE
 will contain 'Some words', and LLINE = 15.
 1...5...10...15...20

**** CHIN ****

Integer function to convert a numeric character string to an integer.

Usage: CHARACTER STRING * (n)
INTEGER CHIN
...
NUMBER = CHIN (STRING)

*** Parameters ***

CHIN (STRING)

STRING - in - ch** - string to be converted

CHIN - out - int - integer value of string

*** Example ***

CHIN (STRING)

CHARACTER LINE * 10
INTEGER CHIN, NUMBER

...
READ '(A)', LINE
NUMBER = CHIN (LINE)
PRINT *, 'The value of ', LINE, ' is', NUMBER

**** CLR_BIT ****

Clear one bit in a bit array (bit string).

Usage: CALL CLR_BIT (BITNO.rl.r, BITS.mv.r)

See also FLP_BIT, SET_BIT, TST_BIT; help module BIT_PKG.

*** Parameters ***

CALL CLR_BIT (BITNO.rl.r, BITS.mv.r)

BITNO - in - int - the number of the bit to be cleared

BITS - i/o - - the bit string or array

*** Example ***

Clear bit 76 in a 100-bit table:

```
INTEGER N_BITS, BITS_WORD, N_WORDS
PARAMETER ( BITS_WORD = 32      ! integer*4 word
N          , N_BITS    = 100    ! in bit array
N          , N_WORDS   = (N_BITS + BITS_WORD - 1) / BITS_WORD
)
INTEGER BITNO, TABLE(N_WORDS)
...
BITNO = 76
CALL CLR_BIT (BITNO, TABLE)
```

*** Admin_info ***

Author: F. Nagy - Fermilab Accelerator Control System

Languages: MACRO

Date written: 01/17/83

Dates revised

**** CPU ****

Get the CPU processor for this node.

usage: CHARACTER * 4 CPU, THIS_CPU

THIS_CPU = CPU ()

*** Parameters ***

CPU ()

CPU - out - ch** - one of: 'V780' (780, 782, or 785)

'V750'

'V730'

'VMIC' (MicroVAX)

*** Examples ***

CHARACTER * 4 CPU, THIS_CPU

THIS_CPU = CPU ()

PRINT *, 'This is running on a ', THIS_CPU, '.'

*** Admin_info ***

Language: DEC VAX/VMS Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 08/21/85

Dates revised

**** CSHUFL ****

Shuffle a character array.

```
Usage:  INTEGER NELTS, SUBARY(NELTS)
        CHARACTER ORIG(NELTS) * (n), REORDR(NELTS) * (n)
        CHARACTER WORK(NELTS) * 1
        ...
        CALL CSHUFL (ORIG, NELTS, REORDR, SUBARY, WORK)
```

See also CSORT, CSORTD; CSORT2, CSORT2D; CSORTN, CSORTND; ISORTC, ISORTCD.

*** Parameters ***

CALL CSHUFL (ORIG, NELTS, REORDR, SUBARY, WORK)

```
ORIG   - in  - ch** - original array to be shuffled
NELTS  - in  - int  - number of elements to be shuffled
REORDR - out - ch** - shuffled array
SUBARY  - out - int  - array to contain the reordered subscripts
                      (the original position of REORDR(i) is
                      ORIG(SUBARY(i)))
WORK    - out - ch*1 - work array
```

*** Example ***

Sort a character array into ascending order.

```
CHARACTER * 4 ORIG(10) / 'AMDS', 'CACR', 'CASG', 'CAWE', 'CASR',
A  'CAKB', 'CABT', 'CAHS', 'CAHB', 'CAMK' /
CHARACTER * 4 REORDR(10), WORK(10)
INTEGER SUBARY(10)
...
CALL CSHUFL (ORIG, 10, REORDR, SUBARY, WORK)
```

After the sort, REORDR will contain the elements of ORIG in a random order. The i-th element of SUBARY will point to the original position of REORDR(i) in the ORIG array, that is, ORIG(SUBARY(i)) = REORDR(i).

**** CSORT ****

Sort (ascending) a character array.

Usage: INTEGER NELTS
 CHARACTER CARRAY(NELTS) * (n), CTEMP * (n)
 ...
 CALL CSORT (CARRAY, NELTS, CTEMP)

See also CSHUFL; CSORTD; CSORT2, CSORT2D; CSORTN, CSORTND; ISORTC, ISORTCD.

*** Parameters ***

CALL CSORT (CARRAY, NELTS, CTEMP)

CARRAY - i/o - ch** - array to be sorted

NELTS - in - int - number of elements to be sorted

CTEMP - out - ch** - variable of the same length as CARRAY,
 used for swapping

*** Example ***

Sort a character array into ascending order.

```

CHARACTER CARRAY(3) * 20 / ! array to be sorted
A          'CASG...', 'AMDS...', 'CACR...' /
CHARACTER WORK      * 20  ! work element for the sort (must be
                          ! at least as large as the length of
                          ! CARRAY)
INTEGER NELTS / 3 /      ! number of records to be sorted
...
CALL CSORT (CARRAY, NELTS, WORK)
```

After the sort, CARRAY will contain 'AMDS...', 'CACR...', 'CASG...'.

**** CSORT2 ****

Sort (ascending) a character array with an associated character array.

```
Usage:  INTEGER NELTS
        CHARACTER CARRAY(NELTS) * (n), CTEMP * (n)
        CHARACTER ASSOC(NELTS) * (m), CTEMPA * (m)
        ...
        CALL CSORT (CARRAY, NELTS, CTEMP, ASSOC, CTEMPA)
```

See also CSHUFL; CSORT, CSORTD; CSORT2D; CSORTN, CSORTND; ISORTC, ISORTCD.

*** Parameters ***

CALL CSORT2 (CARRAY, NELTS, CTEMP, ASSOC, CTEMPA)

CARRAY - i/o - ch** - array to be sorted

NELTS - in - int - number of elements to be sorted

CTEMP - out - ch** - variable of the same length as CARRAY,
used for swapping

ASSOC - i/o - ch** - associated character array which will
be re-ordered to maintain a 1-to-1 corre-
spondence with the elements of CARRAY

CTEMPA - out - ch** - variable of same length as ASSOC, used for swapping

*** Example ***

Sort a character array with an associated character array into ascending order.

```
CHARACTER CARRAY(3) * 20 / ! array to be sorted
A          'CASG...', 'AMDS...', 'CACR...' /
CHARACTER ASSOC(3) * 55 ! associated array
CHARACTER WORK * 55 ! work element for the sort (must be
                    ! at least as large as the maximum of
                    ! the length of CARRAY and the length
                    ! of ASSOC)
INTEGER NELTS / 3 / ! number of records to be sorted
...
CALL CSORT2 (CARRAY, NELTS, WORK, ASSOC, WORK)
```

After the sort, CARRAY will contain 'AMDS...', 'CACR...', 'CASG...'.
ASSOC will contain the corresponding data.

**** CSORT2D ****

Sort (descending) a character array with an associated character array.

Usage: INTEGER NELTS
 CHARACTER CARRAY(NELTS) * (n), CTEMP * (n)
 CHARACTER ASSOC(NELTS) * (m), CTEMPA * (m)
 ...
 CALL CSORT2D (CARRAY, NELTS, CTEMP, ASSOC, CTEMPA)

See also CSHUFL; CSORT, CSORTD; CSORT2; CSORTN, CSORTND; ISORTC, ISORTCD.

*** Parameters ***

CALL CSORT2D (CARRAY, NELTS, CTEMP, ASSOC, CTEMPA)

CARRAY - i/o - ch** - array to be sorted

NELTS - in - int - number of elements to be sorted

CTEMP - out - ch** - variable of the same length as CARRAY
 used for swapping

ASSOC - i/o - ch** - associated character array which will
 be re-ordered to maintain a 1-to-1 corre-
 spondence with the elements of CARRAY

CTEMPA - out - ch** - variable of the same length as ASSOC
 used for swapping

*** Example ***

Sort (descending) a character array with an associated character array.

```

CHARACTER CARRAY(3) * 20 / ! array to be sorted
A          'CASG...', 'AMDS...', 'CACR...' /
CHARACTER ASSOC(3) * 55 ! associated array
CHARACTER WORK      * 55 ! work element for the sort (must be
                        ! at least as large as the maximum of
                        ! the length of CARRAY and the length
                        ! of ASSOC)
INTEGER NELTS / 3 /      ! number of records to be sorted
...
CALL CSORT2D (CARRAY, NELTS, WORK, ASSOC, WORK)
```

After the sort, CARRAY will contain 'CASG...', 'CACR...', 'AMDS...';
 ASSOC will contain the corresponding data.

**** CSORTD ****

Sort (descending) a character array.

```
Usage:  INTEGER NELTS
        CHARACTER CARRAY(NELTS) * (n), CTEMP * (n)
        ...
        CALL CSORTD (CARRAY, NELTS, CTEMP)
```

See also CSHUFL; CSORT; CSORT2, CSORT2D; CSORTN, CSORTND; ISORTC, ISORTCD.

*** Parameters ***

CALL CSORTD (CARRAY, NELTS, CTEMP)

CARRAY - i/o - ch** - array to be sorted

NELTS - in - int - number of elements to be sorted

CTEMP - out - ch** - variable of the same length as CARRAY,
used for swapping

*** Example ***

Sort a character array into descending order.

```
CHARACTER CARRAY(3) * 20 / ! array to be sorted
A          'CASG...', 'AMDS...', 'CACR...' /
CHARACTER WORK      * 20  ! work element for the sort (must be
                          ! at least as large as the length of
                          ! SHORT)
INTEGER NELTS / 3 /      ! number of records to be sorted
...
CALL CSORTD (CARRAY, NELTS, WORK)
```

After the sort, CARRAY will contain 'CASG...', 'CACR...', 'AMDS...'.

***** CSORTN *****

Sort (ascending) a character array having an associated non-character array.

Usage: INTEGER NELTS
 CHARACTER CARRAY(NELTS) * (n), CTEMP * (n)
 <non-character type> ASSOC(NELTS)
 ...
 CALL CSORTN (CARRAY, NELTS, CTEMP, ASSOC)

See also CSHUFL; CSORT, CSORTD; CSORT2, CSORT2D; CSORTND; ISORTC, ISORTCD.

*** Parameters ***

CALL CSORTN (CARRAY, NELTS, CTEMP, ASSOC)

CARRAY - i/o - ch** - array to be sorted

NELTS - in - int - number of elements to be sorted

CTEMP - out - ch** - variable of the same length as CARRAY,
 used for swapping

ASSOC - i/o - - associated non-character array which will
 be re-ordered to maintain a 1-to-1 corre-
 spondence with the elements of CARRAY

*** Example ***

Sort a 3-element character*100 array into ascending order in positions 2-5. An associated integer array contains pointers to the original position in an array.

(This is useful if you have long records to sort on a short field. Instead of sorting the long records, extract the sort field into another array and set the elements of the associated array to 1..n. Then after sorting, the i-th element of the associated array will point to the j-th element of the long record.)

```
CHARACTER LONG(3) * 100 / ! sort characters 2-5
A      '.CASG...', '.AMDS...', '.CACR...' /
CHARACTER SHORT(3) * 4   ! array to hold sort field
CHARACTER WORK          * 4 ! work element for the sort (must be
                             ! at least as large as the length of
                             ! SHORT)
```

```
INTEGER POINTER(3)      ! associated array of pointers
INTEGER N
INTEGER NELTS / 3 /     ! number of records to be sorted
...
DO 110 N=1,NELT
  SHORT(N) = LONG(N)(2:5) ! extract sort field
  POINTER(N) = N          ! set up pointer
110 CONTINUE
CALL CSORTN (SHORT, NELTS, WORK, POINTER)
```

After the sort, SHORT will contain 'AMDS', 'CACR', 'CASG', and
long(pointer(1)) will be the long record for 'AMDS', etc.

***** CSORTND *****

Sort (descending) a character array having an associated non-character array.

```

Usage:      INTEGER NELTS
            CHARACTER CARRAY(NELTS) * (n), CTEMP * (n)
            <non-character type> ASSOC(NELTS)
            ...
            CALL CSORTND (CARRAY, NELTS, CTEMP, ASSOC)

```

See also CSHUFL; CSORT, CSORTD; CSORT2, CSORT2D; CSORTN; ISORTC, ISORTCD.

*** Parameters ***

CALL CSORTND (CARRAY, NELTS, CTEMP, ASSOC)

CARRAY - i/o - ch** - array to be sorted

NELTS - in - int - number of elements to be sorted

CTEMP - out - ch** - variable of the same length as CARRAY,
used for swapping

ASSOC - i/o - - associated non-character array which will be re-ordered to maintain a 1-to-1 correspondence with the elements of CARRAY

*** Example ***

Sort a 3-element character*100 array on positions 2-5. An associated integer array contains pointers to the original position in an array.

(This is useful if you have long records to sort on a short field. Instead of sorting the long records, extract the sort field into another array and set the elements of the associated array to l.n. Then after sorting, the i-th element of the associated array will point to the j-th element of the long record.)

```

CHARACTER LONG(3) * 100 / ! sort characters 2-5
A 'CASG...', 'AMDS...', 'CACR...' /
CHARACTER SHORT(3) * 4 ! array to hold sort field
CHARACTER WORK * 4 ! work element for the sort (must be
! at least as large as the length of

```

```
                                ! SHORT)
INTEGER POINTER(3)             ! associated array of pointers
INTEGER N
INTEGER NELTS / 3 /            ! number of records to be sorted
...
DO 110 N=1,NELT
    SHORT(N) = LONG(N)(2:5) ! extract sort field
    POINTER(N) = N           ! set up pointer
110 CONTINUE
    CALL CSORTND (SHORT, NELTS, WORK, POINTER)
```

After the sort, SHORT will contain 'CASG', 'CACR', 'AMDS', and
LONG(POINTER(1)) will be the long record for 'CASG', etc.

**** DIGIT ****

Test a character for a digit.

Usage: CHARACTER * 1 CH
LOGICAL DIGIT
...
IF (DIGIT(CH)) THEN
...

*** Parameters ***

DIGIT (CH)

CH - in - ch*1 - character to be tested
DIGIT - out - log - TRUE - CH is a digit
FALSE - CH is not a digit

*** Example ***

Read a character string and flag all digits.

```
...
CHARACTER STRING * 50, FLAGS * 50
...
FLAGS = ' '
READ (*, '(A)') STRING
DO 110 N=1,50
  IF (DIGIT (STRING(N:N))) FLAGS(N:N) = '^'
110 CONTINUE
PRINT *, STRING
PRINT *, FLAGS
```

Then, for STRING='abcde FGHIJ kLmnO pQRst UvWxy Z1234567890()\$'
FLAGS = ' '

**** DIGITS ****

Test a character string for digits.

Usage: CHARACTER * (n) STRING
LOGICAL DIGITS
...
IF (DIGITS(STRING)) THEN
...

*** Parameters ***

DIGITS (STRING)

STRING - in - ch** - string to be tested

DIGITS - out - log - TRUE - string was all digits
FALSE - string was not all digits

*** Example ***

Read a character string and test for all digits.

```
...  
CHARACTER STRING * 10  
...  
READ (*, '(A)') STRING  
IF (DIGITS (STRING(N:N))) THEN  
    PRINT *, 'The string is all digits.'  
ELSE  
    PRINT *, 'The string has at least one non-digit.'  
END IF  
...
```

Then for STRING='0123456789', this program segment prints:
The string is all digits.

For STRING='abcde6ghij', this program segment prints:
The string has at least one non-digit.

**** FLP_BIT ****

Flip one bit in a bit array (bit string).

Usage: CALL FLP_BIT (BITNO.rl.r, BITS.mv.r)

See also CLR_BIT, SET_BIT, TST_BIT; help module BIT_PKG.

*** Parameters ***

CALL FLP_BIT (BITNO.rl.r, BITS.mv.r)

BITNO - in - int - the number of the bit to be flipped

BITS - i/o - - the bit string or array

*** Example ***

Flip bit 76 in a 100-bit table:

```
INTEGER N_BITS, BITS_WORD, N_WORDS
PARAMETER ( BITS_WORD = 32 ! integer*4 word
N           , N_BITS   = 100 ! in bit array
N           , N_WORDS  = (N_BITS + BITS_WORD - 1) / BITS_WORD
)
INTEGER BITNO, TABLE(N_WORDS)
...
BITNO = 76
CALL FLP_BIT (BITNO, TABLE)
```

*** Admin_info ***

Author: David V. Sommer - DTNSRDC Code 1892.2

Languages: Fortran 77

Date written: 08/30/85

Dates revised

**** FRSTCH ****

Integer function to return the position of the first non-blank in a character string. If the string is all blanks, 0 (zero) is returned.

Usage: CHARACTER STRING * (n)
INTEGER FRSTCH
...
NCHAR = FRSTCH (STRING)

*** Parameters ***

FRSTCH (STRING)

STRING - in - ch** - string to be examined

FRSTCH - out - int - character position of first non-blank

*** Example ***

CHARACTER LINE * 80
INTEGER FLINE, FRSTCH
...
READ '(A)', LINE
FLINE = FRSTCH (LINE)
PRINT *, 'The line starts in position ', FLINE

**** GETSTR ****

Extract character string according to user-defined criteria.

Usage: CHARACTER INSTR * (n)
 CHARACTER OUTSTR * (n)
 CHARACTER MATCH * (n)
 INTEGER CODE, GETSTR, NCHAR
 ...
 NCHAR = GETSTR (INSTR, OUTSTR, CODE, MATCH)

*** Parameters ***

GETSTR (INSTR, OUTSTR, CODE, MATCH)

INSTR - in - ch** - the input string

OUTSTR - out - ch** - the output string

CODE - in - int - extraction criteria - one of:

- 1 - alphanumeric only
- 1 - alphanumeric and blank
- 2 - alphabetic only
- 2 - alphabetic and blank
- 3 - numeric only
- 3 - numeric and blank
- 4 - numeric and minus ('-')
- 4 - numeric and minus and blank
- 5 - while in <match>
- 5 - while not in <match>
- 6 - skip while in <match>
- 6 - skip while not in <match>

MATCH - in - ch** - string of acceptable characters
 (for <code>=5|6)
 string of unacceptable characters
 (for <code>=-5|-6)
 (Note: For <code>=-4|-3|-2|-1|1|2|3|4,
 use ' ')

GETSTR - out - out - will contain the length of the
 extracted or skipped string -or-
 0 - no string
 -1 - code was invalid

*** Examples ***

GETSTR (IN, OUT, CODE, MATCH)

- 1) Extract 3 strings from a "record". The first string is alphanumeric (7 chars max); the second numeric and '-' (3 chars max); the third

everything left up to next comma, blank, period or right parenthesis.

```
CHARACTER RECORD*80, FIRST*7, SECOND*20, THIRD*80
INTEGER CODE, GETSTR, N1, N2, N3
```

```
...
NEXT = 1
N1 = GETSTR (RECORD(NEXT:), FIRST, 1, ' ')
NEXT = NEXT + N1
N2 = GETSTR (RECORD(NEXT:), SECOND(1:3), 4, ' ')
NEXT = NEXT + N2
N3 = GETSTR (RECORD(NEXT:), THIRD, -5, ', .)')
```

GETSTR (IN, OUT, CODE, MATCH)

2) As example 1, except skip leading blanks for each field.

```
CHARACTER RECORD*80, FIRST*7, SECOND*20, THIRD*80
INTEGER CODE, GETSTR, N1, N2, N3
```

```
...
NEXT = 1
NEXT = NEXT + GETSTR (RECORD(NEXT:), ' ', 6, ' ')
N1 = GETSTR (RECORD(NEXT:), FIRST, 1, ' ')
NEXT = NEXT + N1
NEXT = NEXT + GETSTR (RECORD(NEXT:), ' ', 6, ' ')
N2 = GETSTR (RECORD(NEXT:), SECOND(1:3), 4, ' ')
NEXT = NEXT + N2
NEXT = NEXT + GETSTR (RECORD(NEXT:), ' ', 6, ' ')
N3 = GETSTR (RECORD(NEXT:), , THIRD, -5, ', .)')
```

GETSTR (IN, OUT, CODE, MATCH)

3) Extract 5 comma-separated parameters. Note that the last parameter ends with a blank instead of a comma.

```
CHARACTER*80 RECORD, STR1, STR2, STR3, STR4, STR5
INTEGER CODE, GETSTR, N1, N2, ..., N5
```

```
...
NEXT = 1
N1 = GETSTR (RECORD(NEXT:), STR1, -5, ',')
NEXT = NEXT + N1 + 1
N2 = GETSTR (RECORD(NEXT:), STR2, -5, ',')
...
N5 = GETSTR (RECORD(NEXT:), STR5, -5, ' ')
```

*** Admin_info ***

Author: David V. Sommer - DTNSRDC Code 1892.2

Languages: Fortran 77

86/05/30

VAX

NSRDC

GETSTR

Page 2-33

Date written: 07/12/82

Dates revised

10/04/85 - make ALPHABETIC mean both upper and lower case

**** HMS2S ****

Convert hh:mm:ss to seconds.

Usage: CHARACTER * (n) HMS
INTEGER HMS2S, SEC
...
SEC = HMS2S (HMS)

While this routine is normally used to convert standard-format time (hh:mm:ss), it can handle almost any size time string with the restriction that only digits, minus (only allowed as the first non-blank character), and 0-2 colons (or periods) as separators are allowed. If there are no colons, the entire field is treated as seconds; if there is only one colon, then mm:ss is assumed. Each of the up-to-three subfields may be any reasonable length or omitted (e.g., '1::' is the same as '01:00:00'; whereas, '1' is the same as '00:00:01').

See also S2HMS to convert back to hh:mm:ss format.

*** Parameters ***

HMS2S (HMS)

HMS - in - ch** - character time string to be converted
HMS2S - out - int - time converted to seconds
(If HMS is invalid, MAXINT is returned (see
HELP @NSRDC MAXINT).)

*** Examples ***

1) Convert the current wall clock time to seconds.

CHARACTER NOW * 8
INTEGER HMS2S, SEC
...
CALL TIME (NOW)
SEC = HMS2S (NOW)

2) Subtract 3.5 hours from the current time. Note that there are other ways to do this. This assumes that the current time is after 3:30 am.

CHARACTER NOW * 8, NEWTIM * 8, S2HMS * 8
INTEGER HMS2S
CALL TIME (NOW)
NEWTIM = S2HMS (HMS2S(NOW)-HMS2S('3:30:'))

*** Admin_info ***

Language: Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 05/01/74 (isec)

Dates revised

- 03/18/83 - convert to Fortran 77
- change name from ISEC to HMS2S
- 07/08/85 - implement on VAX/VMS
- allow almost any format input

**** IOSTAT_TEXT ****

Convert the Fortran I/O status code to a message.

```
Usage:  character * (c) code
        character * 1  level
        character * (m) msg
        integer iostat, l_code, l_msg
        open (u, fmt, IOSTAT=iostat,...      -or-      READ (... etc.
        if (iostat .ne. 0) then                ! 0 ==> success
            call iostat_text (iostat, level, code, l_code, msg, l_msg)
            print *, '%programe-' // level // '-' // code(:l_code) //
a          ', ' // msg(:l_msg)
```

*** Parameters ***

```
call iostat_text (iostat, level, code, l_code, msg, l_msg)
```

iostat - in - integer - I/O status from Fortran I/O statement

level - out - char*1 - error level (S, E, F, I, W)

code - out - char** - capitalized abbreviated form of message

l_code - out - integer - length of code

msg - out - char** - text of message

l_msg - out - integer - length of msg

*** Examples ***

If the program name in the main help illustration is MYPROG and a "file not found" condition was encountered during the open, the generated message would be:

```
%MYPROG-E-FILNOTFOU, file not found
```

*** Admin_info ***

Language: DEC VAX/VMS Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 06/12/86

Dates revised

**** ISORTC ****

Sort (ascending) an integer array having an associated character array.

Usage: INTEGER NELTS, IARRAY(NELTS)
 CHARACTER CASSOC(NELTS) * (n), CTEMPA * (n)
 ...
 CALL ISORTC (IARRAY, NELTS, CASSOC, CTEMPA)

See also CSHUFL; CSORT, CSORTD; CSORT2, CSORT2D; CSORTN, CSORTND; ISORTCD.

*** Parameters ***

CALL ISORTC (IARRAY, NELTS, CASSOC, CTEMPA)

IARRAY - i/o - int - array to be sorted

NELTS - in - int - number of elements to be sorted

CASSOC - i/o - ch** - associated character array which will be
 re-ordered to maintain a 1-to-1 corre-
 spondence with the elements of IARRAY

CTEMPA - out - ch** - variable of the same length as CARRAY,
 used for swapping

*** Example ***

Sort a 10-element integer array into ascending order. There is an associated character array.

```

      INTEGER NUM(10) / ! array to be sorted
A      4, 77, 12, 4, 99, 100, 88, 13, 123, -5/
      CHARACTER CH(10) * 23 ! associated character array
      CHARACTER WORK * 23 ! work element for the sort (must be at
                           ! least as large as the length of CH)

      INTEGER N
      INTEGER NELTS / 10 / ! number of records to be sorted
      ...
      CALL ISORTC (NUM, NELTS, WORK, CH)
  
```

After the sort, NUM will contain -5, 4, 4, 12, 13, 77, 88, 99, 100, 123.
 CH(i) keeps its relationship to NUM(i), that is, CH(10) after the sort was
 CH(9) before the sort.

**** ISORTCD ****

Sort (descending) an integer array having an associated character array.

Usage: INTEGER NELTS, IARRAY(NELTS)
 CHARACTER CASSOC(NELTS) * (n), CTEMPA * (n)
 ...
 CALL ISORTCD (IARRAY, NELTS, CASSOC, CTEMPA)

See also CSHUFL; CSORT, CSORTD; CSORT2, CSORT2D; CSORTN, CSORTND; ISORTC.

*** Parameters ***

CALL ISORTCD (IARRAY, NELTS, CASSOC, CTEMPA)

IARRAY - i/o - int - array to be sorted

NELTS - in - int - number of elements to be sorted

CASSOC - i/o - ch** - associated character array which will be
 re-ordered to maintain a 1-to-1 corre-
 spondence with the elements of IARRAY

CTEMPA - out - ch** - variable of the same length as CARRAY,
 used for swapping

*** Example ***

Sort a 10-element integer array into descending order. There is an associated character array.

```

INTEGER NUM(10) / ! array to be sorted
A      4, 77, 12, 4, 99, 100, 88, 13, 123, -5/
CHARACTER CH(10) * 23 ! associated character array
CHARACTER WORK * 23 ! work element for the sort (must be at
                     ! least as large as the length of CH)

INTEGER N
INTEGER NELTS / 10 / ! number of records to be sorted
...
CALL ISORTCD (NUM, NELTS, WORK, CH)

```

After the sort, NUM will contain 123, 100, 99, 88, 77, 13, 12, 4, 4, -5.
 CH(i) keeps its relationship to NUM(i), that is, CH(1) after the sort was
 CH(9) before the sort.

**** ISUM ****

Sum an integer array.

Usage: INTEGER NELTS, IARRAY(NELTS), ISUM, TOTAL

...
TOTAL = ISUM (IARRAY, NELTS)

See also SUM.

*** Parameters ***

ISUM (IARRAY, NELTS)

IARRAY - i/o - int - array to be summed

NELTS - in - int - number of elements to be summed

ISUM - out - int - the sum

*** Example ***

Sum a 10-element integer array.

```
INTEGER NUM(10) / ! array to be summed
A      4, 77, 12, 4, 99, 100, 88, 13, 123, -5/
INTEGER NELTS / 10 / ! number of records to be summed
INTEGER ISUM, TOTAL
...
TOTAL = ISUM (NUM, NELTS)
```

After the call, TOTAL will contain 515.

**** IS_VT100 ****

Determine if output (SYSS\$OUTPUT) is VT-100-compatible.

Usage: LOGICAL IS_VT100, VT100

...
VT100 = IS_VT100 ()

*** Parameters ***

IS_VT100 ()

IS_VT100 - out - log - TRUE - output file is VT-100-compatible
FALSE - output file is not VT-100-compatible

*** Examples ***

LOGICAL IS_VT100

...
IF (IS_VT100 ()) THEN
 <fancy output for a VT-100 terminal>
ELSE
 <regular output for a non-VT-100 terminal>
END IF

*** Admin_info ***

Language: DEC VAX/VMS Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 08/21/85

Dates revised

**** ITRANS ****

Integer function to translate characters according to translate tables you specify in the call.

Usage: CHARACTER STRING * (n1)
 CHARACTER FROM * (n2)
 CHARACTER TO * (n2)
 INTEGER ITRANS, NTRANS
 ...
 NTRANS = ITRANS (STRING, FROM, TO)

*** Parameters ***

ITRANS (STRING, FROM, TO)

- STRING - i/o - ch** - string to be translated
- FROM - in - ch** - string of character to be translated
- TO - in - ch** - string of translation characters
- ITRANS - out - int - will contain one of:
 - +n - the number of characters translated
 - 0 - no translation done
 - 1 - no translation done because LEN(FROM) <> LEN(TO)

Remarks: Each occurrence of FROM(i:i) in string is changed to TO(i:i).

See also subroutine TRANS.

*** Example ***

```
CHARACTER LINE * 20
CHARACTER FROM * 26 / 'abcdefghijklmnopqrstuvwxyz' /
CHARACTER TO * 26 / 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' /
INTEGER ITRANS, NTRANS
...
READ '(A)', LINE
NTRANS = ITRANS (LINE, FROM, TO)
```

Assuming that the line read contains 'John & Mary User', then LINE becomes 'JOHN & MARY USER' and NTRANS = 9.

**** JGDATE ****

Convert any Gregorian date to a relative Julian number or vice versa.

Usage: INTEGER JG, JD, GYEAR, GMONTH, GDAY

...
CALL JGDATE (JG, JD, GYEAR, GMONTH, GDAY)

The relative Julian number corresponding to a Gregorian date is the number of days since 11/24/-4713 (extrapolating the Gregorian calendar).

This subroutine is useful in determining the elapsed number of days between any two calendar dates. It can also be used to find the calendar date so many days from any given date.

*** Parameters ***

CALL JGDATE (JG, JD, GYEAR, GMONTH, GDAY)

JG - in - int - direction of conversion
1 - Gregorian to Relative Julian
2 - Relative Julian to Gregorian

JG=1: JD - out - int - will contain relative Julian number
GYEAR - in - int - Gregorian year (e.g., 1985)
GMONTH - in - int - Gregorian month (1-12)
GDAY - in - int - Gregorian day (1-31)

JG=2: JD - in - int - relative Julian number
GYEAR - out - int - will contain Gregorian year (e.g., 1985)
GMONTH - out - int - will contain Gregorian month (1-12)
GDAY - out - int - will contain Gregorian day (1-31)

*** Example ***

INTEGER JD, GY, GM, GD

...
CALL JDDATE (1, JD, 1985, 2, 25)
JD = JD + 1000
CALL JGDATE (2, JD, GY, GM, GD)

This example will find the date 1000 days from 02/25/85.

**** JP_MODE ****

Get the job/process mode (batch, interactive, network, other, or unknown).

Usage: CHARACTER * 11 JP_MODE, MODE

...
MODE = JP_MODE ()

*** Parameters ***

JP_MODE ()

JP_MODE - out - ch** - one of: 'BATCH', 'INTERACTIVE', 'NETWORK',
'OTHER', or 'UNKNOWN'

*** Examples ***

```
CHARACTER * 11 JP_MODE, MODE
...
MODE = JP+MODE ()
IF (MODE .EQ. 'BATCH') THEN
  <do batch-only stuff>
ELSE IF (MODE .EQ. 'INTERACTIVE') THEN
  <do interactive-only stuff>
ELSE IF (MODE .EQ. 'NETWORK') THEN
  <do network-only stuff>
ELSE
  <do other|unknown-only stuff>
END IF
```

*** Admin_info ***

Language: DEC VAX/VMS Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 08/21/85

Dates revised

**** LEFT ****

Integer function to left-justify a character string. The string is left-justified within itself.

Usage: CHARACTER STRING * (n)
 CHARACTER WORK * (n)
 INTEGER LEFT, LSTRING
 ...
 LSTRING = LEFT (STRING, WORK)

*** Parameters ***

LEFT (STRING, WORK)

STRING - i/o - ch** - string to be left-justified

WORK - - ch** - work variable of len(string)

LEFT - out - int - the position of the last non-blank

*** Example ***

CHARACTER LINE * 80
 CHARACTER WORK * 80
 INTEGER LEFT, LLINE
 ...
 READ '(A)', LINE
 LLEFT = LEFT (LINE, WORK)

If LINE contains ' Some words ', then after left justifying, it
 will contain 'Some words ', and LLINE = 10.
 1...5...10...15...20

**** LO2UP ****

Convert lower case to upper case. Non-alphabetic characters are not changed.

Usage: CHARACTER STRING * (n)

...
CALL LO2UP (STRING)

*** Parameter ***

CALL LO2UP (STRING)

STRING - i/o - ch** - string to be translated in place

*** Examples ***

If STRING contains

'AbCdEfGhIjKlMnOpQrStUvWxYz'

then after CALL LO2UP (STRING), STRING will contain

'ABCDEFGHJKLMNOPQRSTUVWXYZ'

**** LSTCH ****

Integer function to return the position of the last non-blank in a character string. If the string is all blanks, 0 (zero) is returned.

Usage: CHARACTER STRING * (n)
INTEGER LSTCH
...
NCHAR = LSTCH (STRING)

*** Parameters ***

LSTCH (STRING)

STRING - in - ch** - string to be examined

LSTCH - out - int - character position of last non-blank

*** Example ***

CHARACTER LINE * 80
INTEGER LLINE, LSTCH
...
READ '(A)', LINE
LLINE = LSTCH (LINE)
PRINT *, 'The line is ', LLINE, ' characters long.'

**** MAXAI ****

Find the maximum of an array of integers.

Usage: INTEGER ARRAY(n), NELTS, MAXAI, MAX_VALUE

...
MAX_VALUE = MAXAI (ARRAY, NELTS)

See also MAXAR, MINAI, MINAR.

*** Parameters ***

MAXAI (ARRAY, NELTS)

ARRAY - in - int - array of integers to be analyzed

NELTS - in - int - number of elements in array

MAXAI - out - int - the maximum value in array

*** Examples ***

```
PROGRAM TEST
IMPLICIT NONE
INTEGER ARRAY(4) / -23, 0, 473, 472/
INTEGER MAXAI
INTEGER NELTS / 4/
TYPE *, 'The maximum value is ', MAXAI (ARRAY, NELTS)
END
```

This will produce the output:

The maximum value is 473

*** Admin_info ***

Language: Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 07/10/85

Dates revised

**** MAXAR ****

Find the maximum of an array of real numbers.

Usage: INTEGER NELTS
REAL ARRAY(n), MAXAI, MAX_VALUE
...
MAX_VALUE = MAXAR (ARRAY, NELTS)

See also MAXAI, MINAI, MAXAI.

*** Parameters ***

MAXAR (ARRAY, NELTS)

ARRAY - in - real - array of real numbers to be analyzed

NELTS - in - int - number of elements in array

MAXAR - out - real - the maximum value in array

*** Examples ***

```
PROGRAM TEST
IMPLICIT NONE
REAL ARRAY(4) / -23., 0., 473., 472.9/
REAL MAXAR
INTEGER NELTS / 4/
TYPE *, 'The maximum value is ', MAXAR (ARRAY, NELTS)
END
```

This will produce the output:

The maximum value is 473.0000

*** Admin_info ***

Language: Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 07/10/85

Dates revised

**** MAXINT ****

Return the maximum integer supported by VAX/VMS.

Usage: INTEGER MAXINT, VALU

...
VALU = MAXINT ()

See also MININT to obtain the maximum negative integer.

*** Parameter ***

MAXINT ()

MAXINT - out - int - the maximum integer supported by VAX/VMS

*** Example ***

Find the minimum value in an array of integers.

```
INTEGER FUNCTION MIN_ARRAY (ARRAY, N_ARRAY)
INTEGER ARRAY (*), N_ARRAY
INTEGER MAXINT, N
MIN_ARRAY = MAXINT ()
DO N=1,N_ARRAY
  MIN_ARRAY = MIN (MIN_ARRAY, ARRAY(N))
END DO
RETURN
END
```

*** Admin_info ***

Language: Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 07/08/85

Dates revised

*** MAXREAL ***

Return the maximum real number supported by VAX/VMS.

Usage: REAL MAXREAL, VALU

...
VALU = MAXREAL ()

See also MINREAL to obtain the smallest absolute real number.

*** Parameter ***

MAXREAL ()

MAXREAL - out - int - the maximum real number supported by VAX/VMS

*** Example ***

Find the minimum value in an array of real numbers.

```
REAL FUNCTION MIN_ARRAY (ARRAY, N_ARRAY)
REAL ARRAY (*), MIN_ARRAY
INTEGER N, N_ARRAY
MIN_ARRAY = MAXREAL ()
DO N=1,N_ARRAY
  MIN_ARRAY = MIN (MIN_ARRAY, ARRAY(N))
END DO
RETURN
END
```

*** Admin_info ***

Language: Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 07/10/85

Dates revised

**** MFRAME ****

Obtain the machine and node on which the program is running.

Usage: CHARACTER CPU * 10, NODE * 3

...
CALL MFRAME (CPU, NODE)

*** Parameters ***

CALL MFRAME (CPU, NODE)

CPU - out - ch** - the machine (always 'VAXcluster')

NODE - out - ch** - the node ('DT1' or 'DT2')

*** Example ***

CHARACTER CPU * 10, NODE * 3

...
CALL MFRAME (CPU, NODE)
TYPE *, 'This program is running on node ', NODE,
. ' of the ', CPU, '.'

will type: This program is running on node DTn of the VAXcluster.

**** MINAI ****

Find the minimum of an array of integers.

Usage: INTEGER ARRAY(N), nelts, MINAI, MIN_VALUE

MIN_VALUE = MINAI (ARRAY, NELTS)

See also MAXAR, MAXAI, MINAR.

*** Parameters ***

MINAI (ARRAY, NELTS)

ARRAY - in - int - array of integers to be analyzed

NELTS - in - int - number of elements in array

MINAI - out - int - the minimum value in array

*** Examples ***

```
PROGRAM TEST
IMPLICIT NONE
INTEGER ARRAY(4) / -23, 0, 473, 472/
INTEGER MINAI
INTEGER NELTS / 4/
TYPE *, 'The minimum value is ', MINAI (ARRAY, NELTS)
ENL
```

This will produce the output:

The minimum value is -23

*** Admin_info ***

Language: Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 07/10/85

Dates revised

**** MINAR ****

Find the minimum of an array of real numbers.

Usage: INTEGER NELTS
REAL ARRAY(n), MINAI, MIN_VALUE
...
MIN_VALUE = MINAR (ARRAY, NELTS)

See also MAXAI, MAXAR, MINAI.

*** Parameters ***

MINAR (ARRAY, NELTS)

ARRAY - in - real - array of real numbers to be analyzed

NELTS - in - int - number of elements in array

MINAR - out - real - the minimum value in array

*** Examples ***

```
PROGRAM TEST
IMPLICIT NONE
REAL ARRAY(4) / -23., 0., 473., 472.9/
REAL MINAR
INTEGER NELTS / 4/
TYPE *, 'The minimum value is ', MINAR (ARRAY, NELTS)
END
```

This will produce the output:

The minimum value is -23.0000

*** Admin_info ***

Language: Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 07/10/85

Dates revised

**** MININT ****

Return the maximum negative integer supported by VAX/VMS.

Usage: INTEGER MININT, VALU

...
VALU = MININT ()

See also MAXINT to obtain the maximum positive integer.

*** Parameter ***

MININT ()

MININT - out - int - the minimum integer supported by VAX/VMS

*** Example ***

Find the maximum value in an array of integers.

```
INTEGER FUNCTION MAX_ARRAY (ARRAY, N_ARRAY)
INTEGER ARRAY (*), N_ARRAY
INTEGER MININT, N
MAX_ARRAY = MININT ()
DO N=1,N_ARRAY
    MAX_ARRAY = MAX (MAX_ARRAY, ARRAY(N))
END DO
RETURN
END
```

*** Admin_info ***

Language: Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 07/08/85

Dates revised

**** MINREAL ****

Return the minimum real number (absolute value) supported by VAX/VMS.

Usage: REAL MINREAL, VALU

...
VALU = MINREAL ()

See also MAXREAL to obtain the largest absolute real number.

*** Parameter ***

MINREAL ()

MINREAL - out - int - the minimum real number (absolute value)
supported by VAX/VMS

*** Example ***

Find the maximum value in an array of positive, non-zero real numbers.

```
REAL FUNCTION MAX_POS_ARRAY (ARRAY, N_ARRAY)
REAL ARRAY (*), MAX_ARRAY
INTEGER N, N_ARRAY
MAX_ARRAY = MINREAL ()
DO N=1,N_ARRAY
    MAX_ARRAY = MIN (MAX_ARRAY, ARRAY(N))
END DO
RETURN
END
```

*** Admin_info ***

Language: Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 07/10/85

Dates revised

**** MOVEIT ****

Move an array.

Usage: REAL FROM, TO -or- INTEGER FROM, TO
INTEGER NWORDS

...
CALL MOVEIT (FROM, TO, NWORDS)

*** Parameters ***

CALL MOVEIT (FROM, TO, NWORDS)

FROM - in - real/int - array to be moved

TO - out - real/int - output array

NWORDS - in - int - number of words to be moved

*** Examples ***

Save a 100-word integer array A in A_SAVE:

...
INTEGER A(100), A_SAVE(100)

...
CALL MOVEIT (A, A_SAVE, 100)

*** Admin_info ***

Language: Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 10/16/79

Dates revised

**** NARGS ****

In a subprogram, get the number of arguments in the call.

```
Usage:  SUBROUTINE SUB (<args>)
        INTEGER NARGS, NOARGS
        NOARGS = NARGS ()      -or-      CALL NARGS (NOARGS)
        ...
```

*** Parameters ***

```
NOARGS = NARGS ()
CALL NARGS (NOARGS)
```

NARGS - out - int - number of arguments in the actual call to
the subprogram

NOARGS - out - int - same as NARGS

*** Example ***

```
NOARGS = NARGS ()
CALL NARGS (NOARGS)
```

```
PROGRAM TEST
...
CALL SUB (ARG1, ARG2, ARG3)
...
END
SUBROUTINE SUB (A1, A2, A3, A4, A5, A6)
INTEGER NARGS, NOARGS
NOARGS = NARGS ()
TYPE *, 'Called with ', NOARGS, ' arguments.'
...
RETURN
END
```

In this example, the output will be:

Called with 3 arguments.

*** Admin_info ***

Language: MACRO

Author: F. Nagy - Fermilab Accelerator Control System - ACNET

Date written: 06/07/82

Dates revised

06/08/82 - 04/15/83 - 09/02/83 - 10/19/84

08/16/85 - LIB_ removed from start of routine name
- added to NSRDC.OLB at DTNSRDC

**** NEWFILETYPE ****

Replace the file type (and version) of a filespec with a new file type.

Usage: CHARACTER INFYL * (ni), OUTFYL * (no), TYPE * (nt)

...
CALL NEWFILETYPE (INFYL, OUTFYL, TYPE)

*** Parameters ***

CALL NEWFILETYPE (INFYL, OUTFYL, TYPE)

INFYL - in - ch** - the input file specification

OUTFYL - out - ch** - the output file specification with the new
file type field (and no version number)

TYPE - in - ch** - the new file type (without the '.')

*** Example ***

CALL NEWFILETYPE (INFYL, OUTFYL, TYPE)

The output file specification is to be the same as the input, except
that the file type is to be 'LIS':

CHARACTER INFYL * 128, OUTFYL * 128

...
INQUIRE (1, NAME=INFYL)
CALL NEWFILETYPE (INFYL, OUTFYL, 'LIS')

***** PARS *****

Parse a string.

Usage: CHARACTER * (np) PARSCH
 CHARACTER * (n) STRING, PARAM(<maxpar>)
 INTEGER MAXPAR, NPARS, PARS
 ...
 NPARS = PARS (PARSCH, STRING, PARAM, MAXPAR)

See also QUAL_CHAR, QUAL_INT, QUAL_LOG.

*** Parameters ***

PARS (PARSCH, STRING, PARAM, MAXPAR)

PARSCH - in - ch** - delimiter(s)

STRING - in - ch** - character string to be parsed

PARAM - out - ch** - character array to hold the fields

MAXPAR - in - int - maximum number of fields to extract

The delimiters of the fields are PARSCH and a space. When found, PARSCH (if other than a space) is returned as the first character of the field.

*** Example ***

Read a filename and some qualifiers and parse them. The qualifiers start with a slash (/).

```
CHARACTER * 256 STRING, PARAM(10)
INTEGER N, NPARS, PARS
...
TYPE *, 'File?'
ACCEPT 1, STRING
1 FORMAT (A)
NPARS = PARS ('/', STRING, PARAM, 10)
TYPE *, 'npars=', NPARS
DO 110 N=1, NPARS
  TYPE *, 'param(', N, ')=', PARAM(N)
110 CONTINUE
```

If the response to 'File?' is

/NOCC MYFILE/NOSKIP /HEADER /LENGTH=66

then after the call to PARS:

```
NPARS = 5
PARAM(1) = /NOCC
PARAM(2) = MYFILE
PARAM(3) = /NOSKIP
PARAM(4) = /HEADERCC
PARAM(5) = /LENGTH=66
```

(These are the defaults for the AUXPRINT command.)

*** Admin_info ***

Language: VAX/VMS Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 02/06/85

Dates revised

08/08/85 - move to library NSRDC

**** PARSE_FILESPEC ****

Parse a file specification, that is, break it up into it's components.

```
Usage:  CHARACTER * (n) FILESPEC
        INTEGER LFILESPEC
        INTEGER NODE1, NODE2
        INTEGER DEVICE1, DEVICE2
        INTEGER DIRECTORY1, DIRECTORY2
        INTEGER FILENAME1, FILENAME2
        INTEGER FILETYPE1, FILETYPE2
        INTEGER VERSION1, VERSION2

        ...
        CALL PARSE_FILESPEC (FILESPEC , LFILESPEC ,
A                          NODE1      , NODE2      ,
B                          DEVICE1    , DEVICE2    ,
C                          DIRECTORY1 , DIRECTORY2 ,
D                          FILENAME1  , FILENAME2  ,
E                          FILETYPE1  , FILETYPE2  ,
F                          VERSION1   , VERSION2   )
```

This subroutine returns pointers to the beginning and end of each component. For example, FILESPEC(FILETYPE1:FILETYPE2) is the type component. If a component is missing, the pointers are set to zero. The length of the file specification is also returned. No attempt is made to validate the components.

*** Parameters ***

```
CALL PARSE_FILESPEC (FILESPEC , LFILESPEC ,
A                     NODE1      , NODE2      ,
B                     DEVICE1    , DEVICE2    ,
C                     DIRECTORY1 , DIRECTORY2 ,
D                     FILENAME1  , FILENAME2  ,
E                     FILETYPE1  , FILETYPE2  ,
F                     VERSION1   , VERSION2   )
```

FILESPEC - in - ch** - file specification to be parsed

LFILESPEC - out - int - length of filespec

NODE1 - out - int - pointer to start of node

NODE2 - out - int - pointer to end of node

DEVICE1 - out - int - pointer to start of device

DEVICE2 - out - int - pointer to end of device

DIRECTORY1 - out - int - pointer to start of directory

DIRECTORY2 - out - int - pointer to end of directory

FILENAME1 - out - int - pointer to start of file name

FILENAME2 - out - int - pointer to end of file name

FILETYPE1 - out - int - pointer to start of file type
FILETYPE2 - out - int - pointer to end of file type

VERSION1 - out - int - pointer to start of version
VERSION2 - out - int - pointer to end of version

*** Examples ***

```
CALL PARSE_FILESPEC (FILESPEC , LFILESPEC ,  
A      NODE1      , NODE2      ,  
B      DEVICE1    , DEVICE2    ,  
C      DIRECTORY1, DIRECTORY2,  
D      FILENAME1  , FILENAME2  ,  
E      FILETYPE1  , FILETYPE2  ,  
F      VERSION1   , VERSION2   )
```

If filespec contains "MYFILE.TYP", then after the call,
1...5...10

```
LFILESPEC = 10  
NODE1 = NODE2 = DEVICE1 = DEVICE2 = DIRECTORY1 = DIRECTORY2 = 0  
FILENAME1 = 1      FILENAME2 = 6  
FILETYPE1 = 8      FILETYPE2 = 10  
VERSION1 = VERSION2 = 0
```

If filespec contains "USERDISK1:[MYID.JON1234567890]MYFILE.TYP;24",
1...5...10...15...20...25...30...35...40.43

then after the call,

```
LFILESPEC = 43  
NODE1 = NODE2 = 0  
DEVICE1 = 1      DEVICE2 = 10  
DIRECTORY1 = 11  DIRECTORY2 = 30  
FILENAME1 = 31   FILENAME2 = 36  
FILETYPE1 = 38   FILETYPE2 = 40  
VERSION1 = 42    VERSION2 = 43
```

**** QUAL_CHAR ****

Get the value of a character qualifier (/qual=string).

```
Usage:  CHARACTER QUAL_FIELD * (nf), QUAL_VALUE * (nv)
        CHARACTER QUAL_NAME * (nn), DEFAULT * (nd)
        CHARACTER WORK_FIELD * (nf), WORK_NAME * (nn)
        INTEGER MINCH
        LOGICAL QUAL_CHAR
        ...
        IF (QUAL_CHAR (QUAL_NAME, WORK_NAME, MINCH, DEFAULT,
        .             QUAL_FIELD, WORK_FIELD, QUAL_VALUE)) THEN
        ...
```

See also QUAL_INT, QUAL_LOG, PARS.

*** Parameters ***

QUAL_CHAR (QUAL_NAME, WORK_NAME, MINCH, DEFAULT, QUAL_FIELD,
WORK_FIELD, QUAL_VALUE)

```
QUAL_NAME - in - ch** - qualifier name (e.g., '/QUAL')
WORK_NAME - scr - ch** - work variable of length >= LEN(QUAL_NAME)
MINCH      - in - int  - minimum number of characters to be tested
                  (if MINCH=1, then /Q, /QU, /QUA and /QUAL
                  are recognized)
DEFAULT    - in - ch** - default value if only '/QUAL' or '/QUAL='
QUAL_FIELD - in - ch** - field to be checked and evaluated
WORK_FIELD - scr - ch** - work variable of length >= LEN(QUAL_FIELD)
QUAL_VALUE - out - ch** - returned value of '/QUAL=value'
QUAL_CHAR  - out - log  - TRUE  - QUAL_FIELD was QUAL_NAME and a value
                        has been returned
                        FALSE - QUAL_FIELD is not QUAL_NAME and no
                        value is returned
```

*** Example ***

QUAL_CHAR (QUAL_NAME, WORK_NAME, MINCH, DEFAULT, QUAL_FIELD,
WORK_FIELD, QUAL_VALUE)

After extracting the qualifier, see if it is /TYPE=type. If it is,

QUAL_VALUE will contain 'type'.

```
CHARACTER * 15 QUAL_FIELD, QUAL_VALUE, WORK_NAME
LOGICAL QUAL_CHAR
...
IF (QUAL_CHAR ('/TYPE', WORK_NAME, 1, 'deftype', QUAL_FIELD,
&      WORK_NAME, QUAL_VALUE)) THEN
  <the qualifier was TYPE>
ELSE
  <the qualifier was not TYPE>
END if
```

*** Admin_info ***

Language: VAX/VMS Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 03/27/85

Dates revised

05/15/85 - ?

08/07/85 - change name from getqulc to qual_char
- generalize by adding work_name and work_field parameters
- move to library NSRDC

**** QUAL_INT ****

Get the value of an integer qualifier (/qual=integer).

```
Usage:  CHARACTER QUAL_FIELD * (nf), QUAL_NAME * (nn)
        CHARACTER WORK_FIELD * (nf), WORK_NAME * (nn)
        INTEGER DEFAULT, MINCH, QUAL_VALUE
        LOGICAL QUAL_INT
        ...
        IF (QUAL_INT (QUAL_NAME, WORK_NAME, MINCH, DEFAULT,
        .             QUAL_FIELD, WORK_FIELD, QUAL_VALUE)) THEN
        ...
```

See also QUAL_CHAR, QUAL_LOG, PARS.

*** Parameters ***

```
QUAL_INT (QUAL_NAME, WORK_NAME, MINCH, DEFAULT, QUAL_FIELD,
        WORK_FIELD, QUAL_VALUE)

QUAL_NAME  - in  - ch** - qualifier name (e.g., '/QUAL')
WORK_NAME  - scr - ch** - work variable of length >= LEN(QUAL_NAME)
MINCH      - in  - int  - minimum number of characters to be tested
                        (if MINCH=1, then /Q, /QU, /QUA and /QUAL
                        are recognized)
DEFAULT    - in  - int  - default value if only '/QUAL' or '/QUAL='
QUAL_FIELD - in  - ch** - field to be checked and evaluated
WORK_FIELD - scr - ch** - work variable of length >= LEN(QUAL_FIELD)
QUAL_VALUE - out - int  - returned value of '/QUAL=value'
QUAL_INT   - out - log  - TRUE  - QUAL_FIELD was QUAL_NAME and a value
                        has been returned
                        FALSE - QUAL_FIELD is not QUAL_NAME and no
                        value is returned
```

*** Example ***

```
QUAL_INT (QUAL_NAME, WORK_NAME, MINCH, DEFAULT, QUAL_FIELD,
        WORK_FIELD, QUAL_VALUE)
```

After extracting the qualifier, see if it is /LENGTH=length. If it is, QUAL_VALUE will contain <length> as an integer.

```
CHARACTER * 15 QUAL_FIELD, WORK_FIELD
CHARACTER * 7 WORK_NAME
INTEGER QUAL_VALUE
LOGICAL QUAL_INT
...
IF (QUAL_INT ('/LENGTH', WORK_NAME, 1, 66, QUAL_FIELD,
&          WORK_FIELD, QUAL_VALUE)) THEN
  <the qualifier was LENGTH>
ELSE
  <the qualifier was not LENGTH>
END IF
```

*** Admin_info ***

Language: VAX/VMS Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 03/27/85

Dates revised

05/15/85 - ?

08/07/85 - change name from getqulc to qual_char
- generalize by adding work_name and work_field parameters
- move to library NSRDC

**** QUAL_LOG ****

Get the value of an logical qualifier (/qual or /NOqual).

```
Usage:  CHARACTER QUAL_FIELD * (nf), QUAL_NAME * (nn)
        CHARACTER WORK_NAME * (nn+2)
        INTEGER MINCH, QUAL_VALUE
        LOGICAL QUAL_LOG
        ...
        IF (QUAL_LOG (QUAL_NAME, WORK_NAME, MINCH, QUAL_FIELD,
        .             QUAL_VALUE)) THEN
        ...
```

See also QUAL_CHAR, QUAL_INT, PARS.

*** Parameters ***

QUAL_LOG (QUAL_NAME, WORK_NAME, MINCH, NOMINCH, QUAL_FIELD, QUAL_VALUE)

QUAL_NAME - in - ch** - qualifier name (e.g., '/QUAL')

WORK_NAME - scr - ch** - work variable of length >= LEN(QUAL_NAME)+2

MINCH - in - int - minimum number of characters to be tested
(if MINCH=1, then /Q, /QU, /QUA and /QUAL
are recognized)

QUAL_FIELD - in - ch** - field to be checked and evaluated

QUAL_VALUE - out - log - returned value of
TRUE - /qual was found
FALSE - /noqual was found

QUAL_LOG - out - log - TRUE - QUAL_FIELD was QUAL_NAME and a value
has been returned
FALSE - QUAL_FIELD is not QUAL_NAME and no
value is returned

*** Example ***

QUAL_LOG (QUAL_NAME, WORK_NAME, MINCH, NOMINCH, QUAL_FIELD, QUAL_VALUE)

After extracting the qualifier, see if it is /SUPPRESS or /NOSUPPRESS.

```
CHARACTER * 15 QUAL_FIELD
CHARACTER * 11 WORK_NAME
LOGICAL QUAL_LOG, QUAL_VALUE
...
IF (QUAL_LOG ('/SUPPRESS', WORK_NAME, 1, QUAL_FIELD,
```

```
&          QUAL_VALUE)) THEN  
  <the qualifier was /SUPPRESS>  
ELSE  
  <the qualifier was /NOSUPPRESS>  
END IF
```

*** Admin_info ***

Language: VAX/VMS Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 03/27/85

Dates revised

05/15/85 - ?

08/07/85 - change name from GETQULC to QUAL_CHAR

- generalize by adding WORK_NAME and WORK_FIELD parameters

- move to library NSRDC

**** REPLAC ****

Integer function to translate characters into other characters.

Usage: CHARACTER STRING * (n1)
 CHARACTER FROM * (n2)
 CHARACTER TO * (n2)
 INTEGER REPLAC, N_REPLACED
 ...
 N_REPLACED = REPLAC (STRING, FROM, TO)

*** Parameters ***

REPLAC (STRING, FROM, TO)

STRING - i/o - ch** - string to be translated

FROM - in - ch** - string of character to be replaced

TO - in - ch** - string of replacement characters

REPLAC - out - int - will contain one of:

+n - the number of characters replaced

0 - no replacement done

-1 - no replacement done because
 LEN(FROM) <> LEN(TO)

-2 - no replacement done because
 FROM or TO was empty

Each occurrence of FROM(i:i) in string is changed to TO(i:i).

*** Example ***

```
character line * 20
character from * 26 / 'abcdefghijklmnopqrstuvwxyz' /
character to * 26 / 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' /
integer l_line, n_replaced, replac
...
read '(a)', l_line, line
n_replaced = replac (line(:l_line, from, to)
```

Assuming that the line read contains 'John & Mary User', then LINE becomes 'JOHN & MARY USER' and N_REPLACED = 9.

*** Related_commands ***

REPLAC - replace characters by characters

REPLEQ - replace characters by a character

REPLNE - replace non-specified characters by a character

*** Admin_info ***

Language: DEC VAX/VMS Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 02/14/86

Dates revised

**** REPLEQ ****

Integer function to translate characters into a single character.

Usage: CHARACTER STRING * (n1)
 CHARACTER FROM * (n2)
 CHARACTER TO * 1
 INTEGER REPLEQ, N_REPLACED
 ...
 N_REPLACED = REPLEQ (STRING, FROM, TO)

*** Parameters ***

REPLEQ (STRING, FROM, TO)

STRING - i/o - ch** - string to be translated

FROM - in - ch** - string of character to be replaced

TO - in - ch*1 - replacement character

REPLEQ - out - int - will contain one of:

- +n - the number of characters replaced
- 0 - no replacement done
- 1 - no replacement done because
LEN(TO) > 1
- 2 - no replacement done because
FROM or TO was empty

Each occurrence of FROM(i:i) in string is changed to TO.

*** Examples ***

Replace all digits with a minus sign (-):

```
character string * 80
integer n_replaced, repleq
...
n_replaced = repleq (string, '0123456789', '-')

```

*** Related_commands ***

REPLAC - replace characters by characters

REPLEQ - replace characters by a character

REPLNE - replace non-specified characters by a character

*** Admin_info ***

Language: DEC VAX/VMS Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 02/14/86

Dates revised

**** REPLNE ****

Integer function to translate unspecified characters into a single character.

Usage: CHARACTER STRING * (n1)
 CHARACTER FROM * (n2)
 CHARACTER TO * 1
 INTEGER REPLNE, N_REPLACED
 ...
 N_REPLACED = REPLNE (STRING, FROM, TO)

*** Parameters ***

REPLNE (STRING, FROM, TO)

STRING - i/o - ch** - string to be translated
 FROM - in - ch** - string of characters NOT to be replaced
 TO - in - ch*1 - replacement character
 REPLNE - out - int - will contain one of:
 +n - the number of characters replaced
 0 - no replacement done
 -1 - no replacement done because LEN(TO) > 1
 -2 - no replacement done because FROM or TO was empty

Each non-occurrence of FROM(i:i) in string is changed to TO.

*** Examples ***

Replace everything but digits with a blank:

character string * 80
 integer n_replaced, replne
 ...
 n_replaced = replne (string, '0123456789', ' ')

*** Related_commands ***

REPLAC - replace characters by characters
 REPLEQ - replace characters by a character
 REPLNE - replace unspecified characters by a character

86/05/30

VAX

NSRDC

REPLNE

Page 2-76

*** Admin_info ***

Language: DEC VAX/VMS Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 02/14/86

Dates revised

**** REVERSE ****

Subroutine to reverse the order of the characters in a character string.

Usage: CHARACTER STRING * (n)

...
CALL REVERSE (STRING)

*** Parameters ***

CALL REVERSE (STRING)

STRING - i/o - ch** - string to be reversed

*** Example ***

CHARACTER LINE * 26 / 'abcdefghijklmnopqrstuvwxyz' /

...
TYPE *, 'Before: 'LINE
CALL REVERSE (LINE)
TYPE *, 'After: 'LINE

results in the following two lines being typed:

Before: abcdefghijklmnopqrstuvwxyz
After: zyxwvutsrqponmlkjihgfedcba

**** RIGHT ****

Integer function to right-justify a character string. The string is right-justified within itself.

Usage: CHARACTER STRING * (n)
 CHARACTER WORK * (n)
 INTEGER LSTRING, RIGHT
 ...
 LSTRING = RIGHT (STRING, WORK)

*** Parameters ***

CALL RIGHT (STRING, WORK)

STRING - i/o - ch** - string to be right-justified

WORK - - ch** - work variable of len(string)

RIGHT - out - int - the position of the last non-blank

*** Example ***

CHARACTER LINE * 80
 CHARACTER WORK * 80
 INTEGER LLINE, RIGHT
 ...
 READ '(A)', LINE
 LLINE = RIGHT (LINE, WORK)

If LINE contains ' Some words ', then after right justifying, it
 will contain ' Some words', and LLINE = 20.
 1...5...10...15...20

**** S2HMS ****

Convert seconds to hh:mm:ss.

Usage: CHARACTER * (n) HMS, S2HMS
INTEGER SEC

...
HMS = S2HMS (SEC)

(n) must be at least big enough (minimum 8) to hold the complete output.

See also HMS2S to convert back to seconds.

*** Parameters ***

S2HMS (SEC)

SEC - in - int - seconds to be converted

S2HMS - out - ch** - time converted to hh:mm:ss

*** Examples ***

1) Convert seconds to hh:mm:ss.

CHARACTER HMS * 8, S2HMS * 8
INTEGER TIM

...
TIM = 61
HMS = S2HMS (TIM)

HMS will contain '00:01:01'.

2) Subtract 3.5 hours from the current time. Note that there are other ways to do this. This assumes that the current time is after 3:30 am.

CHARACTER NOW * 8, NEWTIM * 8, S2HMS * 8
INTEGER HMS2S
CALL TIME (NOW)
NEWTIM = S2HMS (HMS2S(NOW)-HMS2S('3:30:'))

*** Admin_info ***

Language: Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 05/08/74 (ihms)

Dates revised

- 03/18/83 - convert to Fortran 77
- change name from ihms to s2hms
- 07/08/85 - implement on VAX/VMS
- allow for more than 99 hours
- allow for negative seconds

**** SET_BIT ****

Set one bit in a bit array (bit string).

Usage: CALL SET_BIT (BITNO.rl.r, BITS.mv.r)

See also CLR_BIT, FLP_BIT, TST_BIT; help module BIT_PKG.

*** Parameters ***

CALL SET_BIT (BITNO.rl.r, BITS.mv.r)

BITNO - in - int - the number of the bit to be set

BITS - i/o - - the bit string or array

*** Example ***

Set bit 76 in a 100-bit table:

```

INTEGER N_BITS, BITS_WORD, N_WORDS
PARAMETER ( BITS_WORD = 32      ! integer*4 word
N           , N_BITS    = 100    ! in bit array
N           , N_WORDS   = (N_BITS + BITS_WORD - 1) / BITS_WORD
)
INTEGER BITNO, TABLE(N_WORDS)
...
BITNO = 76
CALL SET_BIT (BITNO, TABLE)

```

*** Admin_info ***

Author: F. Nagy - Fermilab Accelerator Control System

Languages: MACRO

Date written: 01/17/83

Dates revised

**** SIGDIG ****

Return number of significant digits (including 1 for a minus sign, if needed)

Usage: integer n, n_digits, sigdig
n_digits = sigdig (n)

This is useful for left-justifying integers in an output format. Use "I<sigdig(number)>" in the format statement.

NOTE: "number" is only tested for up to 9 significant places (+ 1, if negative). If the absolute value of "number" is greater than this, -1 is returned.

*** Parameters ***

number - in - int - number to be tested

sigdig - out - int - number of significant digits (+ 1 if negative)
(if |number| > 10**8, sigdig = -1)

*** Examples ***

Print the message "The file has <n> records.", where <n> is in the variable N_RECS and, for this example, has the value 123:

```
PRINT *, 'The file has ', n, ' records.'
```

will print "The file has 123 records."

```
PRINT 1, n  
1 FORMAT ('The file has ', I<sigdig(n)>, ' records.')
```

will print "The file has 123 records."

*** Admin_info ***

Language: DEC VAX/VMS Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 11/16/81

Dates revised

06/10/85 - convert to VAXcluster

AD-A173 627

COMPUTER CENTER DEC VAXCLUSTER LIBRARIES/NSRDC
(SUBPROGRAMS)(U) DAVID W TAYLOR NAVAL SHIP RESEARCH AND
DEVELOPMENT CENTER BET D V SOMMER MAY 86

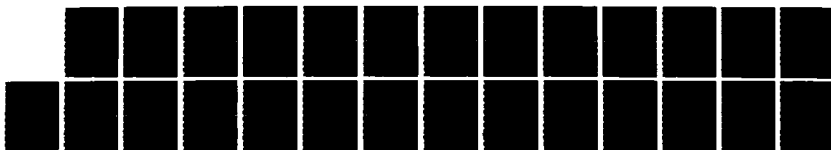
2/2

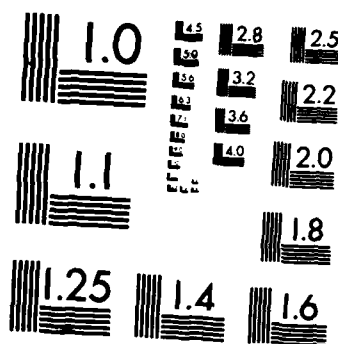
UNCLASSIFIED

DTNSRDC/CMLD-TM-18-86-13

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

**** SUM ****

Sum a real array.

Usage: INTEGER NELTS
 REAL ARRAY(NELTS), sum, total
 ...
 TOTAL = SUM (ARRAY, NELTS)

See also ISUM.

*** Parameters ***

SUM (ARRAY, NELTS)

ARRAY - i/o - real - array to be summed

NELTS - in - int - number of elements to be summed

SUM - out - int - the sum

*** Example ***

Sum a 10-element real array.

REAL NUM(10) / ! array to be summed
A 4., 77., 12., 4., 99., 100., 88., 13., 123., -5./
INTEGER NELTS / 10 / ! number of records to be summed
REAL SUM, TOTAL
...
TOTAL = SUM (NUM, NELTS)

After the call, TOTAL will contain 515.0.

**** SWAPCASE ****

Swap upper and lower case. That is, convert lower case and upper case to upper case and lower case, respectively. Non-alphabetic characters are not changed.

Usage: CHARACTER STRING * (n)

...
CALL SWAPCASE (STRING)

*** Parameter ***

CALL SWAPCASE (STRING)

STRING - i/o - ch** - string to be translated in place

*** Examples ***

If STRING contains

'AbCdEfGhIjKlMnOpQrStUvWxYz'

then after CALL SWAPCASE (STRING), STRING will contain

'aBcDeFgHiJkLmNoPqRsTuVwXyZ'

**** SY ****

Solve a tridiagonal system of equations following the Thomas algorithm.

```
Usage;  INTEGER FIRSUB, LSTSUB
        REAL BEHIND(*), DIAG(*), AHEAD(*), CNSTVC(*)
        ...
        CALL SY (FIRSUB, LSTSUB, BEHIND, DIAG, AHEAD, CNSTVC)
```

*** Parameters ***

CALL SY (FIRSUB, LSTSUB, BEHIND, DIAG, AHEAD, CNSTVC)

FIRSUB - in - int - subscript of first equation

LSTSUB - in - int - subscript of last equation

BEHIND - in - real - coefficient behind of diagonal

DIAG - i/o - real - coefficient on diagonal

AHEAD - in - real - coefficient ahead of diagonal

CNSTVC - i/o - real - element of constant vector
(will contain the solution)

*** Remarks ***

To use this subroutine, the equations must be of the form

$$\begin{bmatrix}
 \overline{D}_{\text{firsub}} & & & & & & & & \\
 & \overline{A}_{\text{firsub}} & & & & & & & \\
 \overline{B}_i & & \overline{D}_i & & \overline{A}_i & & & & \\
 & & & \cdot & & \cdot & & \cdot & \\
 & & & & \cdot & & \cdot & & \cdot \\
 & & & & & \cdot & & \cdot & \\
 & & & & & & \overline{B}_{\text{lstsub}} & & \overline{D}_{\text{lstsub}} \\
 & & & & & & & \overline{U}_{\text{lstsub}} & \\
 & & & & & & & & \overline{C}_{\text{lstsub}}
 \end{bmatrix}$$

The equations in the system are ordered according to the value of the subscript. The variable FIRSUB corresponds to the subscript of the first equation in the system and LSTSUB corresponds to the subscript of the last equation in the system. The number of equations in the system is LSTSUB - FIRSUB + 1. The solution vector U is returned to the calling program in the CNSTVC array. That is, the constant vector CNSTVC is overwritten in the subroutine with the solution. The DIAG array is also altered by the subroutine. AHEAD and BEHIND remain unchanged.

*** Reference ***

"Computational Fluid Mechanics and Heat Transfer", by Dale A. Anderson, John C. Tannehill, Richard H. Pletcher, Hemisphere Publishing Corporation/McGraw-Hill Book Company, pages 549-550 and Chapter 4.

**** TERMINAL ****

For interactive users, get the terminal name.

Usage: CHARACTER * 8 TERM
INTEGER LTERM
...
CALL TERMINAL (TERM, LTERM)

*** Parameters ***

CALL TERMINAL (TERM, LTERM)

TERM - out - ch*8 - the terminal name

LTERM - out - int - the length of term

*** Examples ***

CHARACTER JP_MODE * 11, TERM * 8
INTEGER LTERM
...
IF (JP_MODE () .EQ. 'INTERACTIVE') THEN
CALL TERMINAL (TERM, LTERM)
...
ELSE
...
END IF

*** Admin_info ***

Language: DEC VAX/VMS Fortran 77

Author: David V. Sommer - DTNSRDC Code 1892.2

Date written: 08/21/85

Dates revised

**** TRANS ****

Translate characters according to translate tables you specify in the call.

```
Usage:  CHARACTER STRING * (n1)
        CHARACTER FROM   * (n2)
        CHARACTER TO     * (n2)
        ...
        DATA /FROM      / '<from-characters>'/
        DATA /TO        / '<to-characters>'/
        ...
        CALL TRANS (STRING, FROM, TO)
```

*** Parameters ***

CALL TRANS (STRING, FROM, TO)

STRING - i/o - ch** - string to be translated

FROM - in - ch** - string of character to be translated

TO - in - ch** - string of translation characters

Remarks: Each occurrence of FROM(i:i) in string is changed to TO(i:i).

See also integer function ITRANS.

*** Example ***

```
CHARACTER LINE * 20
CHARACTER FROM * 26 / 'abcdefghijklmnopqrstuvwxyz'/
CHARACTER TO   * 26 / 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'/
...
READ '(A)', LINE
CALL TRANS (LINE, FROM, TO)
```

This example will change lower case letters to upper case.

**** TST_ARG_DFT ****

In a subprogram, test whether a specific argument in the call exists and is not defaulted.

Usage: SUBROUTINE SUB (<args>)
 LOGICAL EXISTS, TST_ARG_DFT
 EXISTS = TST_ARG_DFT (NARG)
 ...

*** Parameters ***

EXISTS = TST_ARG_DFT (NARG)

NARG - in - byte - the argument number to be tested for

TST_ARG_DFT - out - log - TRUE - the narg-th argument is given
 in the outer procedure argument
 list and is not defaulted
 (argument value is non-zero)

FALSE - narg is greater than the number
 of arguments possible
 - the value of the NARG-th
 argument is zero

*** Example ***

EXISTS = TST_ARG_DFT (NARG)

PROGRAM TEST

...

CALL SUB (ARG1, , ARG3)

...

END

SUBROUTINE SUB (A1, A2, A3)

LOGICAL EXISTS, TST_ARG_DFT

IF (TST_ARG_DFT (NARG)) THEN

<code requiring A2>

ELSE

<code not requiring A2>

END IF

RETURN

END

*** Admin_info ***

Language: MACRO

Author: F. Nagy - Fermilab Accelerator Control System - ACNET

Date written: 06/07/82

Dates revised

86/05/30

VAX

NSRDC

TST_ARG_DFT

Page 2-90

06/08/82 - 04/15/83 - 09/02/83 - 10/19/84
08/16/85 - LIB_ removed from routine name
- added to NSRDC.OLB at DTNSRDC

**** TST_BIT ****

Test one bit in a bit array (bit string).

Usage: LOGICAL BIT_SET, TST_BIT
 BIT_SET = TST_BIT (BITNO.rl.r, BITS.mv.r)

See also CLR_BIT, FLP_BIT, SET_BIT; help module BIT_PKG.

*** Parameters ***

TST_BIT (BITNO.rl.r, BITS.mv.r)

BITNO - in - int - the number of the bit to be tested

BITS - i/o - - the bit string or array

TST_BIT - out - log - TRUE - the bit is set
 FALSE - the bit is not set

*** Example ***

Test bit 76 in a 100-bit table and print a message:

```

INTEGER N_BITS, BITS_WORD, N_WORDS
PARAMETER ( BITS_WORD = 32      ! integer*4 word
N           , N_BITS      = 100  ! in bit array
N           , N_WORDS     = (N_BITS + BITS_WORD - 1) / BITS_WORD
)
INTEGER BITNO, TABLE(N_WORDS)
LOGICAL TST_BIT
...
BITNO = 76
IF (TST_BIT (BITNO, TABLE)) THEN
  PRINT *, 'Bit ', BITNO, ' is set.'
ELSE
  PRINT *, 'Bit ', BITNO, ' is not set.'
END IF

```

*** Admin_info ***

Author: F. Nagy - Fermilab Accelerator Control System

Languages: MACRO

Date written: 01/17/83

Dates revised

86/05/30

VAX

NSRDC

TST_BIT

Page 2-92

**** UP2LO ****

Convert upper case to lower case. Non-alphabetic characters are not changed.

Usage: CHARACTER STRING * (n)

...
CALL UP2LO (STRING)

*** Parameter ***

CALL UP2LO (STRING)

STRING - i/o - ch** - string to be translated in place

*** Examples ***

If STRING contains

'AbCdEfGhIjKlMnOpQrStUvWxYz'

then after CALL UP2LO (STRING), STRING will contain

'abcdefghijklmnopqrstuvwxyz'.

***** UPPER *****

Test a character for upper case letter.

```
Usage:      CHARACTER * 1 CH
            LOGICAL UPPER
            ...
            IF (UPPER(CH)) THEN
            ...
```

*** Parameters ***

UPPER (CH)

[illegible]

*** Example ***

Read a character string and flag all upper case letters.

```

...
CHARACTER STRING * 50, FLAGS * 50

...
FLAGS = ' '
READ (*, '(A)') STRING
DO 110 N=1,50
    IF (UPPER (STRING(N:N))) FLAGS(N:N) = '^'
110 CONTINUE
PRINT *, STRING
PRINT *, FLAGS

```

```
Then, for STRING='abcde FGHIJ kLmnO pQRst UvWxy Z1234567890()$
      FLAGS = '      ^^^^^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^
```

**** USERID ****

Obtain the user initials of the job/session running the program.

Usage: CHARACTER ID * 10
INTEGER LID
...
CALL USERID (ID, LID)

*** Parameters ***

CALL USERID (ID, LID)

ID - out - ch** - user initials

LID - out - int - length of ID

*** Example ***

CHARACTER ID * 10
INTEGER LID
...
CALL USERID (ID, LID)
TYPE *, 'Your User ID is ', ID(:LID), ''

**** V2CDAT ****

Convert VMS format date (dd-mmm-yy) to CDC format (mm/dd/yy).

Usage: CHARACTER CDC * 8, VMS * 9

...
CALL V2CDAT (VMS, CDC)

*** Parameters ***

CALL V2CDAT (VMS, CDC)

VMS - in - ch*9 - VMS format date to be converted (dd-mmm-yy)

CDC - out - ch*8 - CDC format converted date (mm/dd/yy)

*** Example ***

CHARACTER CDC * 8, VMS * 9

...
CALL DATE (VMS)
CALL V2CDAT (VMS, CDC)
TYPE *, 'VMS date is ', VMS
TYPE *, 'CDC date is ', CDC

results in the following output:

VMS date is 11-APR-85
CDC date is 04/11/85

**** WEKDAY ****

Determine the day of the week for any Gregorian date from October 15, 1582 thru February 28, 4000.

Usage: CALL WEKDAY (ERR.wl.r, DAY.wl.r, GY.rl.r, GM.rl.r, GD.rl.r)

Dates from January 1, 1582 thru October 14, 1582 and from March 1, 4000 thru December 31, 4000 are not validated.

Method: See IBM Program Description 360D-03.1.004.

*** Parameters ***

CALL WEKDAY (ERR, DAY, GY, GM, GD)

ERR - out - int - return code

0 - no error

1 - at least one of GY, GM, GD out of range

DAY - out - int - return day of week

0 (Sunday) thru 6 (Saturday)

GY - in - int - Gregorian year (e.g., 1985)

GM - in - int - Gregorian month (1-12)

GD - in - int - Gregorian day (1-31)

*** Examples ***

Find the day of the week for 23 September 1985:

```

PROGRAM SAMPLE
IMPLICIT NONE
INTEGER ERR, DAY, GY, GM, GD
CHARACTER WD(0:6) * 9 / 'Sunday', 'Monday', 'Tuesday', 'Wednesday',
a      'Thursday', 'Friday', 'Saturday' /
GY = 1985
GM = 9
GD = 23
CALL WEKDAY (ERR, DAY, GY, GM, GD)
PRINT 3, GM, GD, GY, WD(DAY)
3 FORMAT (I3.2, '/', I2.2, '/', I4, ' is a ', A)
END

```

*** Admin_info ***

Language: Fortran 77

Author: Richard L. Conner - IBM

Date written: 10/15/66

Dates revised

04/26/73 - rewritten in Fortran for CDC 6700 - DVS

09/23/85 - implement on VAXcluster - DVS

***** Index *****

AC	2-2
Parameters	2-2
Example	2-2
ALFA	2-3
Parameters	2-3
Example	2-3
ALFANU	2-4
Parameters	2-4
Example	2-4
ALFANUS	2-5
Parameters	2-5
Example	2-5
ALFAS	2-6
Parameters	2-6
Example	2-6
BANR	2-7
Parameters	2-7
Example	2-7
BANR6	2-8
Parameters	2-8
Example	2-8
BIT_PKG	2-9
CLR_BIT	2-9
Parameters	2-9
Example	2-9
FLP_BIT	2-9
Parameters	2-9
Example	2-10
SET_BIT	2-10
Parameters	2-10
Example	2-10
TST_BIT	2-10
Parameters	2-11
Example	2-11
Admin_info	2-11
C2VDAT	2-13
Parameters	2-13
Example	2-13
CENTER	2-14
Parameters	2-14
Example	2-14

CHIN	2-15
Parameters	2-15
Example	2-15
CLR_BIT	2-16
Parameters	2-16
Example	2-16
Admin_info	2-16
CPU	2-17
Parameters	2-17
Examples	2-17
Admin_info	2-17
CSHUFL	2-18
Parameters	2-18
Example	2-18
CSORT	2-19
Parameters	2-19
Example	2-19
CSORT2	2-20
Parameters	2-20
Example	2-20
CSORT2D	2-21
Parameters	2-21
Example	2-21
CSORTD	2-22
Parameters	2-22
Example	2-22
CSORTN	2-23
Parameters	2-23
Example	2-23
CSORTND	2-25
Parameters	2-25
Example	2-25
DIGIT	2-27
Parameters	2-27
Example	2-27
DIGITS	2-28
Parameters	2-28
Example	2-28
FLP_BIT	2-29
Parameters	2-29
Example	2-29
Admin_info	2-29
FRSTCH	2-30

Parameters	2-30
Example	2-30
GETSTR	2-31
Parameters	2-31
Examples	2-31
Admin_info	2-32
HMS2S	2-34
Parameters	2-34
Examples	2-34
Admin_info	2-35
IOSTAT_TEXT	2-36
Parameters	2-36
Examples	2-36
Admin_info	2-36
ISORTC	2-38
Parameters	2-38
Example	2-38
ISORTCD	2-39
Parameters	2-39
Example	2-39
ISUM	2-40
Parameters	2-40
Example	2-40
IS_VT100	2-41
Parameters	2-41
Examples	2-41
Admin_info	2-41
ITRANS	2-42
Parameters	2-42
Example	2-42
JGDATE	2-43
Parameters	2-43
Example	2-43
JP_MODE	2-44
Parameters	2-44
Examples	2-44
Admin_info	2-44
LEFT	2-45
Parameters	2-45
Example	2-45
LO2UP	2-46
Parameter	2-46
Examples	2-46

LOWER	2-47
Parameters	2-47
Example	2-47
LSTCH	2-48
Parameters	2-48
Example	2-48
MAXAI	2-49
Parameters	2-49
Examples	2-49
Admin_info	2-49
MAXAR	2-50
Parameters	2-50
Examples	2-50
Admin_info	2-50
MAXINT	2-51
Parameter	2-51
Example	2-51
Admin_info	2-51
MAXREAL	2-52
Parameter	2-52
Example	2-52
Admin_info	2-52
MFRAME	2-53
Parameters	2-53
Example	2-53
MINAI	2-54
Parameters	2-54
Examples	2-54
Admin_info	2-54
MINAR	2-55
Parameters	2-55
Examples	2-55
Admin_info	2-55
MININT	2-56
Parameter	2-56
Example	2-56
Admin_info	2-56
MINREAL	2-57
Parameter	2-57
Example	2-57
Admin_info	2-57
MOVEIT	2-58
Parameters	2-58
Examples	2-58

Admin_info	2-58
NARGS	2-59
Parameters	2-59
Example	2-59
Admin_info	2-59
NEWFILETYPE	2-60
Parameters	2-60
Example	2-60
PARS	2-61
Parameters	2-61
Example	2-61
Admin_info	2-62
PARSE_FILESPEC	2-63
Parameters	2-63
Examples	2-64
QUAL_CHAR	2-65
Parameters	2-65
Example	2-65
Admin_info	2-66
QUAL_INT	2-67
Parameters	2-67
Example	2-67
Admin_info	2-68
QUAL_LOG	2-69
Parameters	2-69
Example	2-69
Admin_info	2-70
REPLAC	2-71
Parameters	2-71
Example	2-71
Related_commands	2-71
Admin_info	2-72
REPLEQ	2-73
Parameters	2-73
Examples	2-73
Related_commands	2-73
Admin_info	2-73
REPLNE	2-75
Parameters	2-75
Examples	2-75
Related_commands	2-75
Admin_info	2-76
REVERSE	2-77
Parameters	2-77

Example	2-77
RIGHT	2-78
Parameters	2-78
Example	2-78
S2HMS	2-79
Parameters	2-79
Examples	2-79
Admin_info	2-79
SET_BIT	2-81
Parameters	2-81
Example	2-81
Admin_info	2-81
SIGDIG	2-82
Parameters	2-82
Examples	2-82
Admin_info	2-82
SUM	2-83
Parameters	2-83
Example	2-83
SWAPCASE	2-84
Parameter	2-84
Examples	2-84
SY	2-85
Parameters	2-85
Remarks	2-85
Reference	2-86
TERMINAL	2-87
Parameters	2-87
Examples	2-87
Admin_info	2-87
TRANS	2-88
Parameters	2-88
Example	2-88
TST_ARG_DFT	2-89
Parameters	2-89
Example	2-89
Admin_info	2-89
TST_BIT	2-91
Parameters	2-91
Example	2-91
Admin_info	2-91
UP2LO	2-93
Parameter	2-93

Examples	2-93
UPPER	2-94
Parameters	2-94
Example	2-94
USERID	2-95
Parameters	2-95
Example	2-95
V2CDAT	2-96
Parameters	2-96
Example	2-96
WEKDAY	2-97
Parameters	2-97
Examples	2-97
Admin_info	2-97

Initial Distribution

Copies:

12 Director
Defense Technical Information Center (DTIC)
Cameron Station
Alexandria, Virginia 23314

Center Distribution

Copies:

1	18/1809	Gleissner, G. H.
1	1805	Cuthill, E. H.
2	1809S	
1	182	Camara, A. W.
1	184	Schot, J. W.
1	185	Schaffran, R.
1	187	Zubkoff, M. J.
1	189	Gray, G. R.
1	189.2	
1	189.3	Morris, J.
20	1892.1	Strickland, J. D.
10	1892.2	Sommer, D. V.
1	1892.3	Minor, L. R.
1	1894	
1	1896	Glover, A.
1	1896.2	Dennis, L.
1	522	TIC (C)
1	522.2	TIC (A)
1	93	Patent Counsel

DTNSRDC ISSUES THREE TYPES OF REPORTS

1. DTNSRDC REPORTS, A FORMAL SERIES, CONTAIN INFORMATION OF PERMANENT TECHNICAL VALUE. THEY CARRY A CONSECUTIVE NUMERICAL IDENTIFICATION REGARDLESS OF THEIR CLASSIFICATION OR THE ORIGINATING DEPARTMENT.

2. DEPARTMENTAL REPORTS, A SEMIFORMAL SERIES, CONTAIN INFORMATION OF A PRELIMINARY, TEMPORARY, OR PROPRIETARY NATURE OR OF LIMITED INTEREST OR SIGNIFICANCE. THEY CARRY A DEPARTMENTAL ALPHANUMERICAL IDENTIFICATION.

3. TECHNICAL MEMORANDA, AN INFORMAL SERIES, CONTAIN TECHNICAL DOCUMENTATION OF LIMITED USE AND INTEREST. THEY ARE PRIMARILY WORKING PAPERS INTENDED FOR INTERNAL USE. THEY CARRY AN IDENTIFYING NUMBER WHICH INDICATES THEIR TYPE AND THE NUMERICAL CODE OF THE ORIGINATING DEPARTMENT. ANY DISTRIBUTION OUTSIDE DTNSRDC MUST BE APPROVED BY THE HEAD OF THE ORIGINATING DEPARTMENT ON A CASE-BY-CASE BASIS.

END

12-86

DTIC